VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTÉM PRO SPRÁVU ZAŘÍZENÍ ROUTEROS

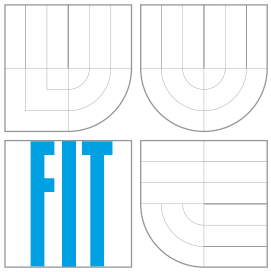
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB KONEČNÝ

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SYSTÉM PRO SPRÁVU ZAŘÍZENÍ ROUTEROS

ROUTEROS DEVICES MANAGEMENT SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAKUB KONEČNÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. JOSEF HÁJEK
SUPERVISOR

BRNO 2012

Abstrakt

Tato bakalářská práce popisuje návrh a realizaci informačního systému pro správu a vzdálenou konfiguraci zařízení se systémem RouterOS od firmy MikroTik. Jeho hlavním účelem je možnost spravovat všechna zařízení na jednom místě. U jednotlivých zařízení lze upravovat nastavení jejich DHCP a DNS protokolů, spravovat statické DNS záznamy a statické směrování. Výsledná aplikace je implementována v jazyce PHP s využitím Nette Framework a databáze MySQL jako úložiště dat.

Abstract

This bachelor's thesis describes a design and an implementation of an information system for management control and remote configuration of devices running MikroTik's RouterOS system. Primary purpose of the system is managing all compatible devices at one place. Within each device you can modify its DNS and DHCP settings, create or edit static DNS records and static routes. System is written in PHP using Nette Framework and MySQL database.

Klíčová slova

MikroTik, RouterOS, RouterBoard, směrovač, informační systém, API, Nette Framework, PHP, MySQL

Keywords

MikroTik, RouterOS, RouterBoard, router, information system, API, Nette Framework, PHP, MySQL

Citace

Jakub Konečný: Systém pro správu zařízení RouterOS, bakalářská práce, Brno, FIT VUT v Brně, 2012

Systém pro správu zařízení RouterOS

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně, pod vedením pana Ing. Josefa Hájka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jakub Konečný
10. května 2012

Poděkování

Rád bych poděkoval vedoucímu své bakalářské práce, Ing. Josefu Hájkovi, za odbornou pomoc a vstřícný přístup při tvorbě této bakalářské práce. Terezo, tobě mnohokrát díky za korekce...

Obsah

1 Úvod **3**

2 MikroTik **4**
2.1 Základní informace 4
2.2 RouterBOARD 5

3 MikroTik RouterOS **6**
3.1 Operační systém RouterOS 6
3.2 Licencování ROS 6
3.3 Konfigurace a správa 7
3.3.1 Konzole, Telnet a SSH 7
3.3.2 Aplikace WinBox 8
3.3.3 Webová rozhraní Webbox a WebFig 9

4 RouterOS API **11**
4.1 Základní informace a použití API 11
4.2 Popis protokolu 11
4.2.1 Příkazy (commands) 12
4.2.2 Argumenty (arguments) 12
4.2.3 Dotazy (queries) 13
4.2.4 Odpovědi (replies) 13
4.2.5 Značení odchozích příkazů (tags) 14
4.3 Autentizační mechanismus 14
4.4 API v jazyce PHP 15
4.4.1 Demonstrace použití třídy 15

5 Analýza a návrh systému **17**
5.1 Koncepce systému 17
5.2 Požadovaná funkcionalita 17
5.3 Návrh ukládání dat 19
5.4 Zajištění konzistence dat 19

6 Implementace systému **21**
6.1 Adresářová struktura 21
6.2 Použité technologie 22
6.3 Architektura MVC 23
6.4 Mechanismus přihlašování a řízení přístupu 23
6.4.1 Autentizace uživatelů 23

6.4.2 Řízení přístupu přes ACL . . . 24
6.4.3 Nastavení oprávnění . . . 24
6.5 Správa uživatelů (UsersPresenter) . . . 24
6.5.1 UsersModel . . . 25
6.6 Podpůrné databázové funkce (UtilityModel) . . . 25
6.7 Práce s daty koncových zařízení (MikrotikModel) . . . 25
6.8 Systémová nástěnka (HomepagePresenter) . . . 26
6.9 Obsluha chybových stavů (ErrorPresenter) . . . 26
6.10 Přihlašování uživatelů (SignPresenter) . . . 27
6.11 Autorizace uživatelů (SecuredPresenter) . . . 27
6.12 Nastavení systému (SettingsPresenter) . . . 28
6.12.1 SettingsModel . . . 28
6.13 Záznam činnosti uživatelů (LogPresenter) . . . 29
6.13.1 LogModel . . . 30
6.14 Správa zařízení (DevicesPresenter) . . . 30
6.15 Statické směrování (RoutingPresenter) . . . 31
6.16 Práce s DNS (DnsPresenter) . . . 32
6.17 Správa DHCP (DhcpPresenter) . . . 33
6.18 Zálohování zařízení (BackupsPresenter) . . . 34
6.18.1 BackupsModel . . . 35

7 Závěr **36**
7.1 Demonstrace vytvořeného systému . . . 36
7.2 Zhodnocení výsledků . . . 37
7.3 Vylepšení do budoucna . . . 38

A Obsah přiloženého CD **41**

B Návod na instalaci **42**

C Metriky kódu **44**

Kapitola 1

Úvod

Tato práce si klade za cíl vytvořit informační systém, pomocí něhož by mohli správci sítě pohodlně a na jednom místě evidovat koncová zařízení, postavená na operačním systému MikroTik RouterOS. Pomocí vyvíjeného nástroje u nich bude také možné konfigurovat vybranou funkcionalitu.

Celá technická zpráva se skládá z celkem sedmi kapitol. Ve druhé kapitole je čtenáři krátce přiblížena firma MikroTik, její filosofie a produkty, které s touto prací souvisí. Na to je navázáno v kapitole 3, zabývající se podrobným popisem operačního systému RouterOS. Na jeho konfiguraci je tato bakalářská práce zaměřena, proto je nezbytné důkladně objasnit jeho možnosti, klíčové vlastnosti a možnosti konfigurace. V souvislosti s RouterOS je čtenáři v kapitole 4 detailně popsáno a vysvětleno dostupné API [1] a práce s ním. Navrhovaný systém konfiguruje konfiguruje koncová zařízení právě pomocí tohoto rozhraní.

Předmětem kapitoly 5 je analýza požadavků, kladených na výslednou aplikaci a navržení její výsledné podoby. Předposlední kapitola blíže objasňuje implementaci výsledného systému, některé vzniklé problémy či jeho klíčové vlastnosti. V závěrečné části je pak vzniklý informační systém demonstrován, zhodnocen a jsou diskutovány možnosti jeho budoucího rozšíření.

[1] *Application Programming Interface* (API) označuje rozhraní pro programování aplikací.

Kapitola 2

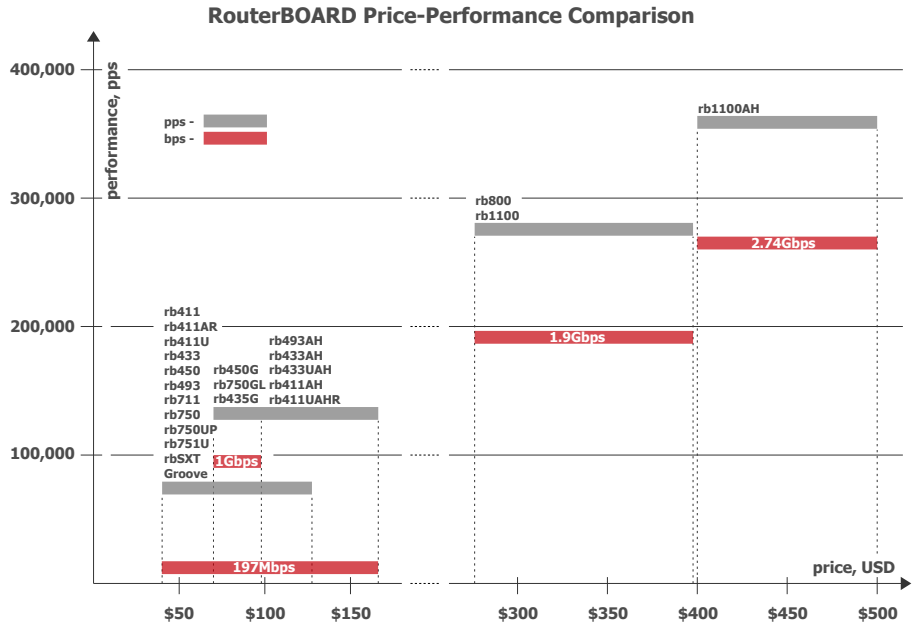
MikroTik

Kapitola je věnována krátkému seznámení s firmou MikroTik. Jedna z podkapitol se zabývá jejím specializovaným síťovým hardwarem, známým též pod označením RouterBOARD. Podrobně jsou popsány možnosti jeho konfigurací a rozdíly mezi nimi.

2.1 Základní informace

Firma MikroTik byla založena roku 1995, nyní sídlí v lotyšském hlavním městě Rize. Od počátku se firma zabývala vývojem směrovačů a bezdrátových systémů pro poskytovatele internetových služeb. Později do svého portfolia přidala operační systém RouterOS a od roku 2002 vyrábí také vlastní hardware (tzv. RouterBOARD) a příslušenství k němu.

Mezi zákazníky firmy MikroTik patří i velcí hráči z průmyslu či výzkumu, jako jsou například Hewlett-Packard, NASA či národní laboratoř v Los Alamos [1]. Jejich produkty si ale našly oblibu i mezi českými poskytovateli internetu. Poměr cena/výkon, pro který jsou zařízení RouterBOARD oceňována, je vidět na obrázku 2.1.

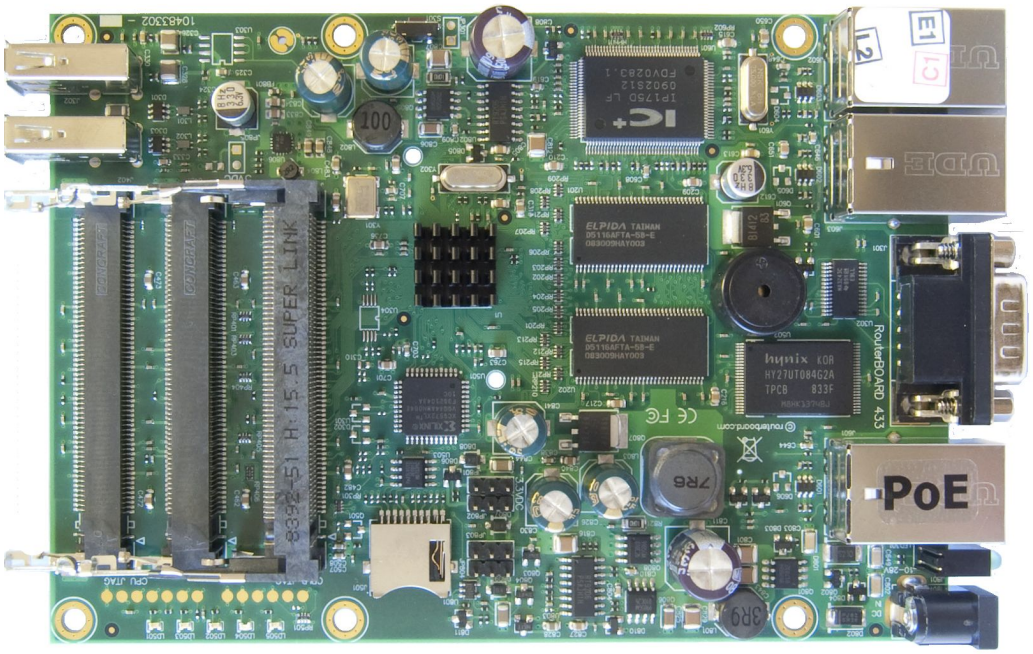


Obrázek 2.1: Srovnání poměru cena/výkon jednotlivých modelů RouterBOARD [2].

2.2 RouterBOARD

RouterBOARD (dále jen RB) je síťový hardware, vyráběný firmou MikroTik. Jedná se o základní desku s integrovaným procesorem, operační pamětí a často také dalšími komponentami. Zařízení jsou postavena na architekturách PowerPC a MIPS-BE. Procesory těchto zařízení patří do rodiny RISC. Jeden z RB si může čtenář prohlédnout na obrázku 2.2.

V závislosti na konkrétním modelu je přítomna různá konektorová výbava. Nejčastěji se lze setkat se sériovým rozhraním zastoupeným konektorem RS232 a RJ45 pro připojení ethernetu. Vyšší modely podporují připojení externích úložišť přes USB, taktéž je možné použít paměťových karet CompactFlash nebo microSD.



Obrázek 2.2: RouterBOARD – model RB433UAH [3].

Samozřejmostí je podpora bezdrátových sítí. Adaptér pro přístup k nim je buď integrovaný nebo ho lze připojit přes sloty miniPCI, respektive miniPCI-E. Výměna nefunkční či nedostačující karty je tak možná bez nutnosti měnit ostatní hardware. Podporovány jsou sítě podle standardů *IEEE 802.11a/b/g/n*. Vyrábějí se také RB se čtečkami SIM karet. Tyto modely lze použít s modemy pro 3G sítě.

RB bývají nejčastěji používány v kombinaci s operačním systémem RouterOS od téhož výrobce. Tato kombinace dělá výrobky MikroTik propracované a výkonné. Lze je provozovat v mnoha režimech, například jako switch, router, případně přístupový bod. Operační systém je čtenáři podrobněji popsán v kapitole 3.

Jednotlivé modely a modelové řady se od sebe liší zejména výkonem procesoru, množstvím operační paměti a počtem ethernetových portů, respektive počtem slotů pro připojení bezdrátových adaptérů.

Kapitola 3

MikroTik RouterOS

Předmětem následující kapitoly je seznámit čtenáře s operačním systémem z dílny společnosti MikroTik. Popsány jsou jeho základní i pokročilé možnosti a způsob jeho licencování. Druhá polovina kapitoly je věnována několika hlavním způsobům konfigurace zařízení s tímto specializovaným OS.

3.1 Operační systém RouterOS

RouterOS (dále jen ROS) je operační systém určený pro síťová zařízení. Je postaven na linuxovém jádře verze 2.6 a jeho vznik se datuje do roku 1997. Systém se vyznačuje zejména širokou škálou dostupných funkcí, svou stabilitou a pohodlnou konfigurací. Verze, dostupná dnes, podporuje ethernet až do rychlosti 10Gbit/s, 3G sítě a bezdrátové sítě podle *IEEE 802.11a/b/g/n*.

Nejčastěji se lze s ROS setkat v zařízeních téhož výrobce. Systém podporuje ale také architekturu x86, což ho umožňuje nainstalovat do téměř jakéholiv počítače. Výhodou poté může být vyšší výkon nebo rozšiřitelnost ve srovnání s jednoúčelovým hardwarem, ovšem za cenu vyšší spotřeby.

ROS nepodporuje běh uživatelských aplikací. Od čtvrté verze ROS je ale možné použít skriptování, postavené na jazyce Lua. Správci tak mohou automatizovat některé úkony či doplnit chybějící funkcionalitu. Jednotky záložního napájení, podporující *APC Smart Protocol*, lze připojit přes sériové rozhraní nebo USB a monitorovat je z prostředí ROS [4]. Pevné disky nebo flash paměti mohou být použity pro ukládání systémových logů nebo jako mezipaměť pro načítané webové stránky.

3.2 Licencování ROS

ROS není otevřeným softwarem, tudíž je jeho používání zatíženo licencí. Zařízení RouterBOARD jsou dodávána s příslušnou licencí, která se odvíjí od konkrétního modelu. Pro počítače na platformě x86 je licenci nutné zakoupit. Licence má neomezenou platnost, je však svázána s konkrétním zařízením a tudíž není přenosná.

Licence ROS jsou rozděleny na stupně, přičemž v současné době jich existuje šest. První dva (*Level0* a *Level1*) lze získat bezplatně a slouží jako demoverze. Licence *Level3* slouží pro bezdrátové klientské stanice a je k dispozici pouze velkoodběratelům. Poslední tři stupně (*Level4*, *Level5* a *Level6*) je možné volně zakoupit. Mezi sebou se liší dostupností, omezeními

některých funkcí, počáteční technickou podporou a pochopitelně cenou. V případě potřeby vyššího stupně je nutné zakoupit novou licenci, protože upgrade na není možný.

Přehled licenčních stupňů a v nich obsažených vlastností je shrnut v tabulce 3.1.

Level	**0 FREE**	**1 Demo**	**3 WISP**	**4 WISP**	**5 WISP**	**6 Controller**
Upgradable to	—	—	ROS v6.x	ROS v6.x	ROS v7.x	ROS v7.x
Initial Config Support	—	—	—	15 dní	30 dní	30 dní
Wireless AP	24h limit	—	—	ano	ano	ano
Wireless Client and Bridge	24h limit	—	ano	ano	ano	ano
RIP, OSPF, BGP protocols	24h limit	—	jen na RB	ano	ano	ano
EoIP tunnels	24h limit	1	neomezené	neomezené	neomezené	neomezené
PPPoE tunnels	24h limit	1	200	200	500	neomezené
PPTP tunnels	24h limit	1	200	200	500	neomezené
L2TP tunnels	24h limit	1	200	200	500	neomezené
OVPN tunnels	24h limit	1	200	200	neomezené	neomezené
VLAN interfaces	24h limit	1	neomezené	neomezené	neomezené	neomezené
HotSpot active users	24h limit	1	1	200	500	neomezené
RADIUS client	24h limit	—	ano	ano	ano	ano
Queues	24h limit	1	neomezené	neomezené	neomezené	neomezené
Web proxy	24h limit	—	ano	ano	ano	ano
Synchronous interfaces	24h limit	—	—	ano	ano	ano
User manager active sessions	24h limit	1	10	20	50	neomezené

Tabulka 3.1: Přehled licencí RouterOS a rozdílů mezi nimi (převzato z [5]).

3.3 Konfigurace a správa

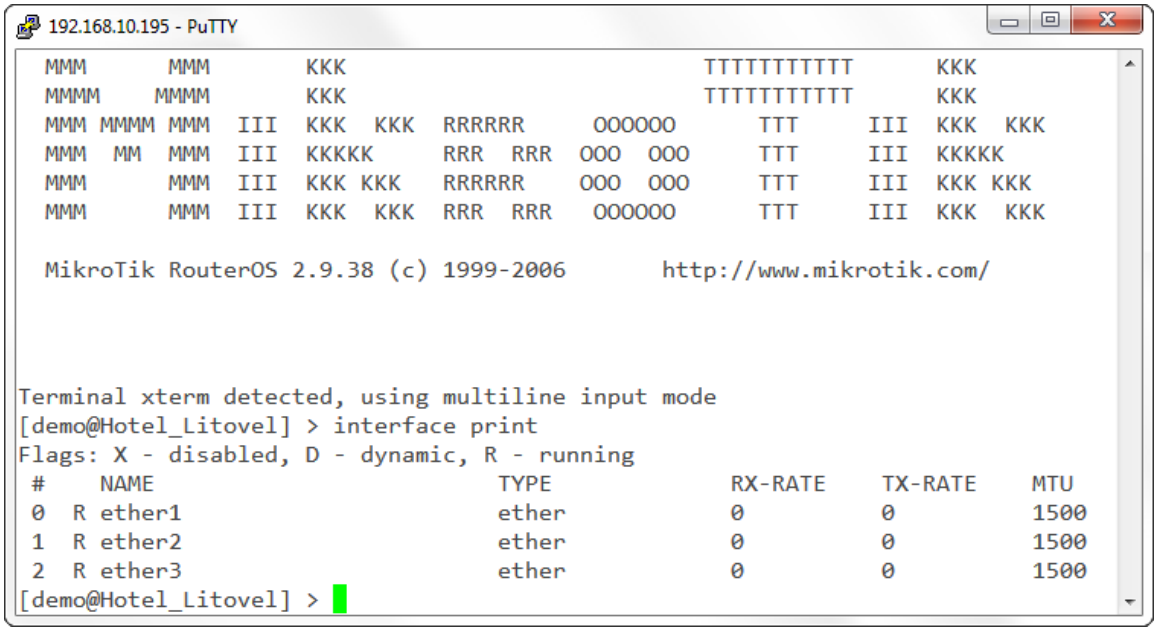
Systém lze konfigurovat různými způsoby, lokálně nebo vzdáleně. Dále jsou čtenáři představeny ty nejpoužívanější způsoby, vyjma použití API. Tomu je věnována samostatná kapitola 4 na straně 11.

Pro všechny způsoby platí, že veškerá provedená nastavení lze snadno vyexportovat do souboru a použít na jiném zařízení. Výměna vadného zařízení je díky tomu rychlá.

3.3.1 Konzole, Telnet a SSH

Konfigurace přes Telnet, SSH a konzoli je velmi podobná. Liší se jen způsobem připojení k zařízení. Pro konzolové spojení se využívá sériového *null-modem* kabelu. To konzoli před-

určuje zejména pro prvotní nastavení nového zařízení nebo k servisním zásahům v případě různých potíží. V obou ostatní případech se jedná o spojení síťové. Připojení přes SSH je z důvodu šifrovaného přenosu dat samozřejmě bezpečnější.



Obrázek 3.1: Terminál při konfiguraci přes konzoli, Telnet či SSH.

Po připojení se zobrazí terminál (viz obrázek 3.1), který slouží pro přímé zadávání konfiguračních příkazů či jejich zkratek. Jednotlivé skupiny nastavení (rozhraní, firewall a další) tvoří stromovou strukturu. Hlavní nabídka má označení `/`, do ostatních skupin se lze přepnout zadáním jejich názvu. Stiskem klávesy `?` dojde k zobrazení nápovědy pro aktuální nabídku. Rozepsané příkazy lze doplňovat stiskem klávesy `Tab`. Pokud existuje jediná shoda, je automaticky doplněna, jinak je vypsán seznam všech odpovídajích příkazů.

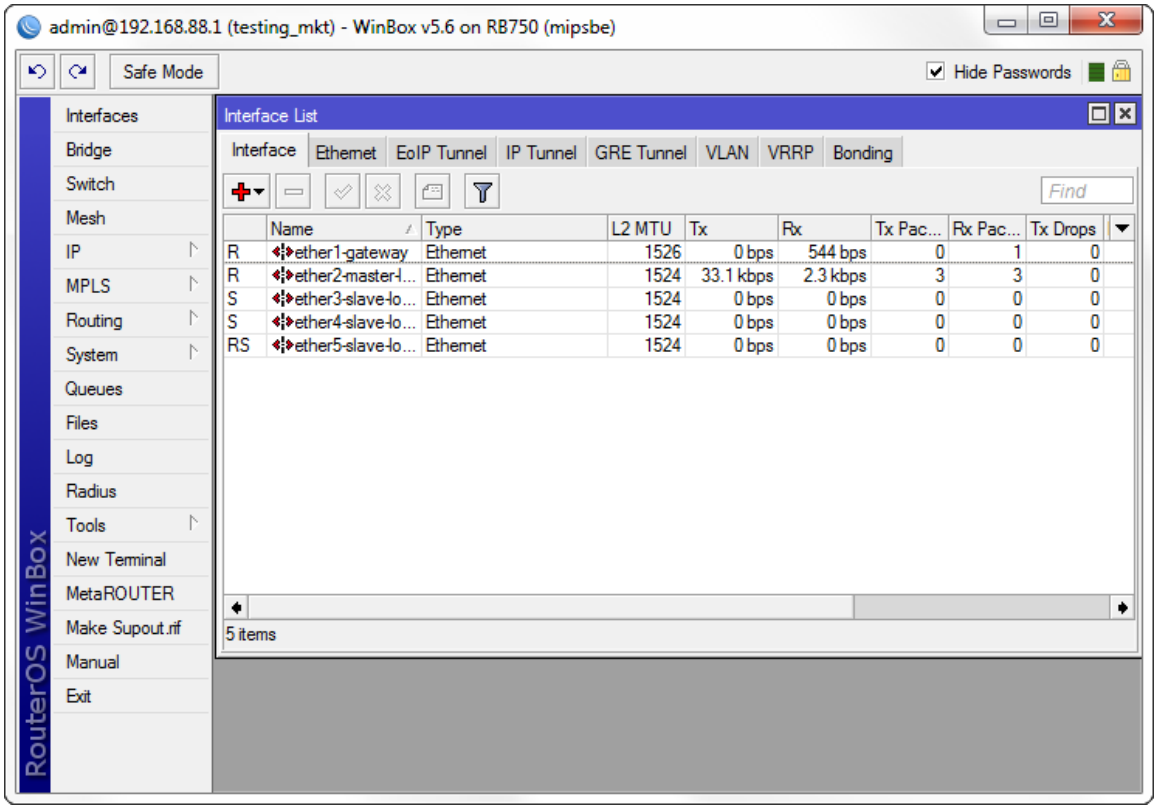
Stejné ovládání sdílí také terminál ve webovém rozhraní WebFig a v aplikaci WinBox.

3.3.2 Aplikace WinBox

WinBox je aplikace pro konfiguraci ROS zařízení skrze grafické uživatelské rozhraní (na obrázku 3.2). Jedná se o nativní aplikaci pro operační systém Microsoft Windows, kterou je možné přes emulátory používat i na systémech Linux či Mac OS X. Slouží k základnímu i pokročilému nastavení ROS, s výjimkou některých kritických operací (např. změna MAC adres síťových rozhraní). Součástí je také terminál pro přímé zadávání konfiguračních příkazů.

Komunikace je možná přes IP protokol, od ROS verze 5.x je implementována i podpora pro IPv6. Je-li zařízení ve stejné podsíti, lze se připojit také přes linkovou vrstvu [1], na základě MAC adresy jeho síťového rozhraní. WinBox umí podporovaná zařízení také vyhledávat s pomocí protokolů *MNDP* (MikroTik Neighbor Discovery Protocol) a *CDP* (Cisco Discovery Protocol).

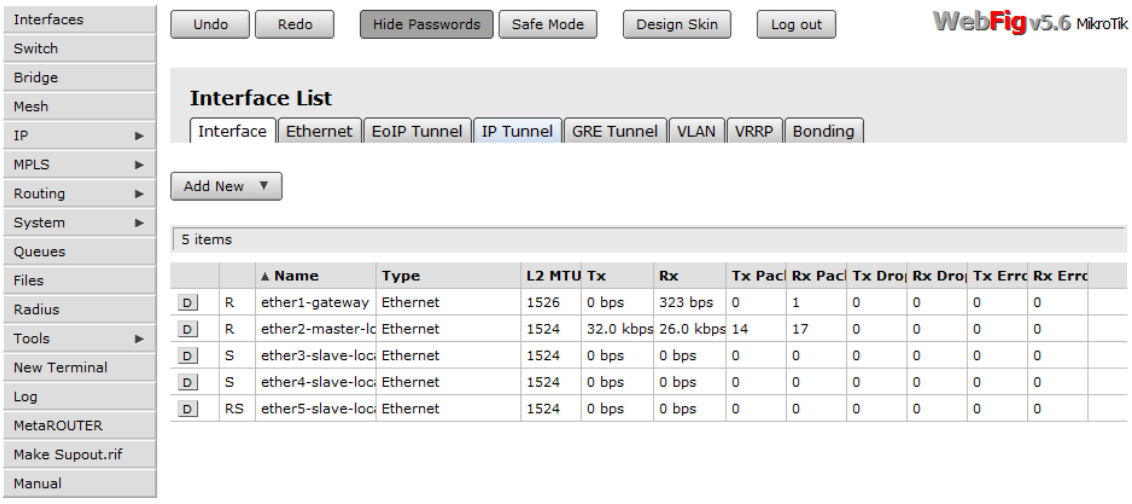
[1] Druhá vrstva referenčního modelu ISO/OSI.



Obrázek 3.2: Prostřední aplikace WinBox.

3.3.3 Webová rozhraní Webbox a WebFig

Zadáním IP adresy zařízení do internetového prohlížeče se otevře webové rozhraní Webbox. Nevýhodou jsou omezené možnosti konfigurace, umožňující jen správu základních funkcí.



Obrázek 3.3: Webové rozhraní WebFig.

Počínaje pátou generací ROS byl Webbox nahrazen komplexnějším rozhraním WebFig, které si lze prohlédnout na obrázku 3.3. Přestože WebFig stále nedosahuje komfortu SSH

či WinBox, jedná se o použitelnou alternativu. Rozhraní je svou strukturou i ovládáním podobné právě výše zmíněné aplikaci. Konfigurovat lze základní ale i pokročilé funkce. K dispozici je také terminál pro přímé zadávání příkazů.

Kapitola 4

RouterOS API

Tato kapitola je věnována přiblížení API, dostupného v systémech RouterOS od verze 3. Na začátku je API popsáno obecně a je vysvětlena motivace jeho možného použití. Dále je popsán protokol, který je při komunikaci použít a všechny jeho klíčové součásti. V závěru kapitoly je představena třída, sloužící pro komunikaci s API v jazyku PHP. Součástí je také komentovaný příklad práce s ní.

4.1 Základní informace a použití API

API je jednou z možností vzdálené správy ROS zařízení. Poskytuje rozhraní například pro tvorbu vlastního software, jež může zahrnovat pouze omezenou funkcionalitu ve srovnání s programem WinBox či lépe implementovat funkce s ohledem na vlastní potřeby.

Služba poskytující API je implicitně zakázaná a je nutné ji povolit příkazem 4.1 nebo v odpovídající nabídce v programu WinBox. Služba poté naslouchá na portu 8728. Ten je možné změnit, stejně jako rozsah IP adres, pro které bude API dostupné.

```
/ip service enable api
```

Ukázka 4.1: Povolení služby API přes terminál.

Přes API lze ovládat většinu funkcí systému. Výjimkou jsou interaktivní části (například telnet nebo SSH klient) a některá omezení, kladená na uživatelské skripty. Není zde také dostupný příkaz `find`, místo něj je možné data filtrovat dotazováním. Jejich mechanismus je popsán v jedné z dalších částí této kapitoly.

Dokumentace obsahuje také ukázky zdrojových kódů pro práci s API v mnoha programovacích jazycích. Příkladem mohou být jazyky C, Java, PHP a mnohé jiné. Autory těchto kódů jsou však sami uživatelé ROS, nejedná se tedy o oficiální materiály výrobce. Z této skutečnosti plyne, že jejich použití je dáno kvalitou implementace a licencí, pod kterou jsou dostupné.

Informace zde uvedené, vycházejí z oficiální API dokumentace, dostupné na [6, 7].

4.2 Popis protokolu

Služba API funguje na principu klient – server, respektive požadavek – odpověď. Požadavky jsou na straně zařízení (serveru) vyřizovány paralelně. Rozpoznat, zdali se jedná o odchozí

či příchozí zprávu, je možné podle jejího prvního řádku. Protokol API je formátován jako posloupnost slov. Každé slovo je zakódováno podle své délky.

Jedno či více slov tvoří datovou větu, která musí být ukončena slovem nulové délky. Ukončení je nutné i v případě věty o délce jednoho slova, jinak by mohlo dojít k nesprávné interpretaci další komunikace. Věta může obsahovat nejvýše jeden příkaz, počet argumentů omezen není. Požadavky jsou zpracovány až po obdržení ukončujícího slova, prázdné věty jsou ignorovány.

Souhrn používaných typů datových vět je předmětem několika následujících podkapitol. V závěru kapitoly je popsán také mechanismus značení datových vět.

4.2.1 Příkazy (commands)

Práce s příkazy v prostředí API jsou velmi podobné, jako při práci s terminálem. Jejich názvy jsou stejné, jako v případě terminálu. Většina příkazů začíná znakem `/`, mezery které se v nich vyskytují jsou nahrazovány tímtéž znakem – viz. ukázka 4.2. Existují také výjimky, o nichž ještě bude řeč.

```
/system/package/getall
```

Ukázka 4.2: Příkaz pro vypsání seznamu systémových balíčků.

Dále existuje pět příkazů, stručně popsaných v následujícím seznamu. Tyto jsou dostupné pouze v API. Bližší informace jsou poté v dokumentaci [6].

1. `/login` slouží pro zahájení komunikace a přihlášení klienta k cílovému zařízení. Autentizací se podrobněji zabývá podkapitola 4.3.

2. `/cancel` umožňuje ukončení běžící operace. Je-li odeslán s argumentem `tag`, ukončí jeden konkrétní příkaz. Jinak přeruší všechny probíhající příkazy, kromě sebe samotného.

3. `listen` se používá pro dlouhodobé sledování některých příkazů. Příkladem mohou být statistiky či přehled přihlášených uživatelů. Příkaz probíhá, dokud není ukončen příkazem z předchozího bodu.

4. `print` je obdoba stejnojmenné funkce v konzoli, avšak s několika drobnými rozdíly. Data nelze filtrovat klauzulí *where*, ale je nutné použít dotazování (viz. podkapitola 4.2.3).

5. `getall` je v současné době již jen alias pro `print`.

4.2.2 Argumenty (arguments)

Argumenty najdou uplatnění v případě, kdy je nutné podrobněji specifikovat použitý příkaz. Lze se s nimi setkat také v odpovědích na klientovy požadavky. Následují bezprostředně po příkazu, respektive hlavičce odpovědi a na jejich pořadí nezáleží. Příklad jejich použití je součástí ukázky 4.7.

Slovo s argumentem začíná znakem `=`, následuje jeho název společně s dalším znakem `=` a jeho hodnota. Ta může být i prázdná, případně obsahovat znak rovnítka. Existují také argumenty specifické pro API, jejichž název začíná tečkou (například `.id`).

Patří mezi ně i `.proplist`, používaný s příkazem `print`. Jedná se o seznam vlastností, které mají být zahrnuty v odpovědi. Není-li použit, jsou do odpovědi zahrnuty všechny vlastnosti, které jsou pro daný objekt dostupné. Hrozí tak zhoršení odezvy zařízení a obecně má jeho nepoužívání negativní vliv na výkon. Ukázka 4.3 krátce demonstruje jeho použití.

```
/system/resource/print
=.proplist=uptime,cpu-load
```

Ukázka 4.3: Příkaz s argumentem `.proplist`.

4.2.3 Dotazy (queries)

Smyslem použití dotazů (v originále *queries*) je umožnit klientské straně filtrování či vyhledávání dat, získaných skrze API. Každý dotaz je vyhodnocován počínaje prvním slovem, přičemž záleží na pořadí jednotlivých slov ve větě. V současné době jsou dotazy podporovány pouze v kombinaci s příkazem `print`.

Způsob zápisu dotazů je podobný argumentům, s tím rozdílem, že slovo s dotazem začíná znakem otazníku. Dále pokračuje názvem argumentu, znakem `=` a hodnotou, na kterou se bude argument testovat či jaká se bude vyhledávat. Tato základní syntaxe se v případě některých podmínek mírně liší. Možnosti dotazů jsou poměrně široké, kromě výše uvedeného nechybí ani relační operátory nebo logické operace. Podrobný přehled je k dispozici opět na [6].

```
/interface/print
?type=ether
?type=vlan
?#|
```

Ukázka 4.4: Filtrování dat s pomocí dotazů.

Na ukázce 4.4 je dotaz se dvěma argumenty, na něž je aplikována logická funkce OR. Výsledkem je seznam všech ethernetových a VLAN rozhraní daného zařízení.

4.2.4 Odpovědi (replies)

První slovo odpovědi začíná znakem `!`, následuje typ odpovědi a nakonec příchozí užitečná data. Výsledkem každého příkazu je za normálních okolností alespoň jedna odpověď. API v RouterOS rozeznává čtyři typy odpovědí. Příklad příchozí odpovědi je součástí ukázky 4.5.

```
!done
.tag=3
```

Ukázka 4.5: Příklad odpovědi na ukázku 4.7.

Zpráva s prvním slovem `!done` značí konec odpovědi a tedy i provádění požadovaného příkazu. Počáteční slovo `!re` plní funkci hlavičky odpovědi a oznamuje, že se jedná o odpověď s užitečnými daty. Zjednodušeně se dá říci, že data přenášená odpovědí, se nachází právě mezi těmito dvěma slovy.

Poslední dva typy odpovědí jsou používány pro signalizaci chybových stavů. Obecné chyby a výjimečné situace jsou uvozeny slovem `!trap`. Na ukázce 4.6 je vidět výsledek vytvoření nového rozhraní s duplicitním názvem. Dojde-li k chybám, z nichž se nelze zotavit, je protistraně zaslána zpráva `!fatal` s podrobnostmi a následně je ukončeno spojení se zařízením. S touto zprávou se lze setkat v důsledku chyb při autentizaci klientské strany.

```
!trap
=category=4
=message=already have device with such name
!done
```

Ukázka 4.6: Odpověď signalizující chybu.

4.2.5 Značení odchozích příkazů (tags)

Jak již dříve zaznělo, API podporuje souběžné provádění více příkazů, aniž by bylo nutné čekat na dokončení předchozího. Zároveň však není zaručeno, že odpovědi dorazí ve stejném pořadí jako odeslané požadavky. Proto API obsahuje značkování odchozích požadavků, umožňující párování požadavků a odpovědí na ně.

Obsahuje-li odeslaný požadavek volitelný parametr `.tag` s nenulovou hodnotou, odpověď na něj obsahuje stejnojmenný parametr a jeho hodnota se neliší od té v požadavku. Jeho použití je součástí ukázky 4.7. Další z jeho funkcí je možnost zrušit nedokončený příkaz na základě jeho značky.

```
/interface/set
=disabled=yes
=.id=ether1
.tag=3
```

Ukázka 4.7: Použití parametru `.tag`.

4.3 Autentizační mechanismus

Přihlašování k cílovému zařízení je dvoufázové a je postaveno na znalosti uživatelského jména a hesla. Samotné ověření uživatelovy identity je poté založeno na kontrole vypočteného autentizačního řetězce. K jeho výpočtu je třeba znát i *token*, který klientovi zašle cílové zařízení. Přímo heslo tak sítí neputuje v otevřené, ani v jakékoliv jiné podobě.

Následující postup popisuje jednotlivé kroky přihlašování. První fázi, vyžádání tokenu, odpovídají kroky 1 a 2. Kroky 3 až 5 poté představují fázi samotného přihlášení.

1. Klientská strana (dále K) pošle cílovému zařízení (dále Z) požadavek s příkazem `/login`.
2. Z) zašle klientovi odpověď s argumentem `ret`. K si tuto hodnotu zapamatuje.
3. K vypočítá autentizační řetězec, jako $00 + md5(\backslash 0 + heslo + ret)$ a tuto hodnotu si zapamatuje.
   - Znak `+` značí konkatenaci řetězců.

- Operace `md5()` představuje výpočet otisku algoritmem MD5.

4. *K* pošle *Z* požadavek s příkazem `/login`. Doplní ho o argumenty `name` (hodnotou je přihlašovací jméno) a `response` (hodnotou je řetězec získaný v předchozím bodě).

5. *Z* potvrdí *K* úspěšné přihlášení, případně ho informuje o proběhlé chybě standardním způsobem (viz. podkapitola 4.2.4).

4.4 API v jazyce PHP

Práce s ROS API v jazyce PHP je možná s pomocí třídy, jejímž hlavním autorem je Denis Basta. Její zdrojový kód [8] je k dispozici k volnému použití. Třída umožňuje snadnou konfiguraci a obousměrnou komunikaci s cílovým zařízením. Nechybí jednoduchý ladící režim, během kterého dochází k podrobnému výpisu prováděných akcí. V následujících odstavcích jsou stručně popsány nejdůležitější metody této třídy.

Připojení k zařízení realizuje metoda `connect`. Samotné spojení je realizováno pomocí *socketů*, tedy přes protokoly transportní vrstvy. Pro připojení je nutné znát cílovou IP adresu a pro ověření identity poté uživatelské jméno a heslo. Nepodaří-li se z jakýchkoliv důvodů spojit se zařízením, následuje krátká prodleva a pokus o spojení se opakuje. Pokud i opakované pokusy končí neúspěchem, je vrácena chyba. Její podrobný popis je uložen do atributu `$error_str`. Úspěšně navázané spojení je udržováno, dokud nedojde k jeho uzavření zavoláním metody `disconnect`.

Odesílání dat probíhá voláním metody `write`. Podporováno je odeslání více požadavků za sebou a tedy i jejich označování (viz. část 4.2.5). Opačnou funkci plní `read`, která přijímá odpověď ze zařízení. Vrácená data jsou následně zpracována a vrácena formou dvourozměrného, asociativního pole. Poslední metoda, sloužící pro komunikaci se zařízením, je `comm` a spojuje obě výše uvedené funkce. Díky ní lze snadno odesílat příkazy, jejichž součástí jsou argumenty. Ty je nutné předat v druhém parametru v podobě asociativního pole. Funkce se sama postará o jejich zpracování a výsledek vrátí stejnou formou, jako `read`.

Pro potřeby výsledné aplikace bude třída upravena. Dojde k lepšímu zapouzdření jejich datových atributů a k doplnění metod pro bezpečnou práci s nimi, podle zvyklostí OOP.

4.4.1 Demonstrace použití třídy

Součástí ukázky 4.8 je krátký zdrojový text, demonstrující práci s API v jazyku PHP. Ukázka staví na dříve představené třídě a smyslem je čtenáři přiblížit její použití. Blíže je příklad okomentován v následujících odstavcích.

Nejprve je nutné vložit zdrojový text třídy pro práci s API a vytvořit její novou instanci. Před započetím komunikace je možné upravit nastavení nového objektu. Zde je pro ilustraci upraven port, na kterém běží API služba. Řádek 7 ukazuje vytvoření spojení s koncovým zařízením (serverem), včetně autentizace klienta. Poté, co je spojení úspěšně navázáno, je odeslán klientský požadavek.

Požadavek zde odesílá metoda `write`, avšak nic nebrání použití `comm`. Smyslem parametru `false` je zabránit předčasnému ukončení datové věty. Volání metody `read` na řádku 14 zajišťuje přečtení odpovědi serveru a její zpracování do podoby dvourozměrného asociativního pole. Použitím parametru `false` ke zpracování nedochází a metoda vrátí odpověď tak, jak ji přijala. Spojení se serverem je nakonec uzavřeno voláním metody `disconnect` na řádku 17. Vestavěná funkce `die` (řádek 20) zde supluje obsluhu chybového stavu. Ta

může nastat při potížích s připojením k cílovému zařízení, případně není-li povolená služba zajišťující API (více viz. 4.1).

```php
<?php
  require('routeros_api.class.php');

  $api = new RouterOS_API();
  $api->setPort(9999);

  $connection_status = $api->connect('192.168.1.254', 'apiuser', '1234');
  if ($connection_status) {
    $api->write('/interface/getall', false);
    $api->write('=.proplist=name,type', false);
    $api->write('?name=ether1-gateway', false);
    $api->write('?#!');

    $result = $api->read();
    print_r($result);

    $api->disconnect();
  }
  else {
    die('Nepodařilo se připojit k cílovému zařízení.');
  }
```

Ukázka 4.8: Práce s RouterOS API v jazyce PHP.

Ukázka tak výstižně demonstruje možnost protokolu API. Výsledkem provedení ukázkového kódu je seznam všech rozhraní zařízení, kromě `ether1-gateway`, jak je uvedeno na řádcích 11 a 12. V odpovědi jsou vráceny pouze jejich typy a názvy, čehož je dosaženo použitím argumentu `.proplist`. Výhodou je také nižší zatížení zařízení, než při jeho absenci.

Kapitola 5

Analýza a návrh systému

Následující kapitola analyzuje specifikaci požadavků kladených na výsledný systém a popisuje funkcionalitu, která se od nich dále odvíjí. S ohledem na požadavky je také navržen způsob ukládání zpracovávaných dat v databázi. Nakonec je diskutován problém konzistence dat vzhledem k jejich synchronizaci mezi centrální databází a spravovanými zařízeními.

5.1 Koncepce systému

Vyvíjený informační systém je koncipovaný do podoby webové aplikace. Ta poběží na vyhrazeném internetovém serveru a prostřednictvím protokolu HTTP bude komunikovat s uživatelem. Komunikace se spravovanými koncovými zařízeními bude probíhat přes API, které poskytuje systém RouterOS.

Výhodou webové aplikace je její nenáročnost na programové vybavení. Pro její použití není třeba instalovat dodatečný software, postačí internetový prohlížeč. Ten je dnes součástí snad každého operačního systému. Celý systém je pak také možné ovládat z mobilních zařízení, jakými jsou chytré telefony či dnes stále oblíbenější tablety.

5.2 Požadovaná funkcionalita

Navrhovaný systém je určen především správcům počítačových sítí, případně pracovníkům technické podpory. Jedná se o interní systém v podobě webové aplikace. Od toho se odvíjí veškeré požadavky na jeho funkcionalitu, zabezpečení a řízení přístupu uživatelů k jednotlivým částem. Cílem je aplikace obsahující často používané funkce, nikoliv konkurenci nástrojům *WinBox* či *Webbox*. Základní konfigurace aplikace, zejména údaje pro spojení s databázovým serverem, je uchována v konfiguračním souboru (více v části 6.1). Nastavení systému, související s jeho během a chováním, jsou poté administrátorům k dispozici přímo přes webové rozhraní. Tato nastavení jsou uchovávána v databázi.

Správa systému poskytuje základní prostředky pro správu uživatelských účtů. Dostupný je přehled existujících uživatelů a skupin, stejně jako detail každého z nich. Jednotlivé účty lze přidávat, měnit a mazat. Přidána je také možnost zablokování účtu, při němž nedojde k jeho vymazání, avšak dotyčnému uživateli je znemožněno přihlášení. O každém uživateli jsou uchovávány základní informace jako uživatelské jméno, e-mail, heslo a skupina, do které přísluší. Volitelně poté další kontaktní či doplňující informace. Každý účet je chráněn uživatelem zvoleným heslem. Uživatelé jsou rozděleni do skupin, přičemž každý je členem právě jedné skupiny. Řízení přístupu probíhá přes ACL (*Access Control List*). Každá

skupina má pevně definovány akce, k nimž mají její členové přístup povolen, respektive zakázán.

Další skupina funkcí se týká obsluhy a nastavení směrování. S ohledem na pokyny vedoucího práce je podporováno pouze směrování statické. Systém umožňuje uživateli přidávat nové cesty pro směrování, stejně jako jiné konfigurační kanály RouterOS a to včetně některých doplňujících informací. Dále je možné prohlížet ty existující a podle potřeby je měnit či odstraňovat. Dynamicky vytvořené cesty či ty se speciálním významem jsou při úpravách automaticky ignorovány. Lze je pouze prohlížet a nehrozí tak nechtěné poškození některého důležitého nastavení.

V požadavcích je dále zahrnuta možnost práce se systémem DNS. Jeho funkcionalita v zařízeních MikroTik je sama o sobě velmi zjednodušená. Jedná se o rekurzivní DNS server s možností lokálních statických záznamů [9]. Systém bude umožňovat nastavit používané DNS servery, základní parametry cache a vlastnosti DNS jako celku. Chybět nebude ani práce s lokálními DNS záznamy, tedy možnost jejich procházení a základních operací s nimi. Administrátor je tak bude moci přidávat, měnit či odstraňovat stávající. Implementována bude také možnost správy vyrovnávací paměti lokálního DNS serveru. Záznamy bude možné nejen prohlížet, ale v případě potřeby také odstraňovat.

Vzhledem k tomu, že protokol DHCP [10] je v prostředí počítačových sítí hojně používaný, jsou možnosti práce s ním v prostředí systému RouterOS poměrně bohaté. S ohledem na účely a cílovou skupinu navrhovaného systému jsou možnosti práce s tímto protokolem cíleny na správu DHCP serverů. Základem je vytváření samotných serverů, úpravy či mazání již existujících. Samozřejmostí je také možnost prohlížení již nakonfigurovaných serverů v daném zařízení. Pro každý server musí být možné nastavit rozsah jím přidělovaných adres, síťovou masku a výchozí bránu, případně další doplňující služby (DNS servery, NTP servery a další). Uživatelé systému mohou také prohlížet přehled aktuálně přidělených adres a případně do nich ručně zasahovat – například je zrušit. Podporována je také statická alokace IP adres. Díky ní je zaručeno, že rozhraní se specifikovanou MAC adresou obdrží od DHCP serveru vždy stejnou IP adresu. Toto mapování lze provádět ručně zadáním související dvojice MAC adresy a IP adresy nebo zjednodušeně také přímo z přehledu aktuálně přidělených adres.

Součástí systému je také funkcionalita pro vytváření záznamů o aktivitě uživatelů a akcích, které provedly. Každá významná akce (například vytvoření nového zařízení, úpravy nastavení či smazání uživatele) musí být zaznamenána do databáze společně se základními údaji o ní. Musí být jasné, který uživatel ji vyvolal, kdy se tak stalo a v kterém místě aplikace k tomu došlo. Smyslem této funkce je umožnit správcům systému odhalovat problémy, vzniklé vinou nesprávné manipulace s nastavením koncových zařízení či úmyslnou sabotáž ze strany některého z uživatelů. Záznamy bude možné zpětně prohlížet, k dispozici budou také nástroje pro základní filtrování zobrazených dat.

Aplikace bude umožňovat správcům sítě práci se zálohami spravovaných zařízení. Pro každé je možné jich vytvářet libovolné množství a podle potřeby je také mazat a prohlížet. Každá záloha obsahuje informace o tom, kdo ji vytvořil a ke kterému zařízení v síti se vztahuje. Volitelně je možné přidat také doplňující poznámky autora, jako například popis zálohy. Záložní data jsou získána příkazem `/export`, který provede kompletní export nastavení. Tato funkcionalita není přímo vyžadována zadáním, avšak vzhledem k její užitečnost byla nad rámec zadání zapracována.

Všechna zařízení, s kterými se v systému pracuje lze zobrazovat ve stručném přehledu a provádět s nimi základní evidenční operace – přidávat nová, měnit či mazat stávající. Pro každou jednotku jsou k dispozici doplňující informace, jako například informace o jejím

hardware, nainstalovaném systému a dostupné licenci. Je také možné vypsat seznam nainstalovaných součástí a tyto libovolně povolovat či zakazovat. Tyto doplňující informace jsou automaticky načteny po přidání zařízení do systému. Zároveň tak dochází k ověření, že spojení správně funguje. Z doplňujících funkcí zmíním již jen možnost vzdáleného restartu zařízení.

5.3 Návrh ukládání dat

Data, s nimiž webová aplikace pracuje, jsou ukládána do databáze. Volba padla na databázi *MySQL* [11], která je dostatečně rozšířená a poskytuje potřebnou množinu funkcí. Model databáze na obrázku 5.1 byl vytvořen na základě rozboru jednotlivých funkcí systému v části 5.2. Znázorňuje uchovávaná data, vztahy mezi nimi a slouží jako základ pro vytvoření modelu a struktury výsledné databáze.

Uchovávána jsou zejména data, potřebná pro správnou funkcionalitu popsanou výše. Výjimku tvoří dynamické záznamy v koncovém zařízení, jako například obsah vyrovnávací paměti systému DNS. Ty jsou v případě nutnosti jednorázově staženy, zobrazeny a dále se s nimi nepracuje. Je také nutné ukládat některé provozní informace a další podpůrná data. Řešení problému konzistence dat, s nimiž se pracuje je popsáno v části 5.4. V aktuální podobě systému neexistuje vztah mezi uživatelskými skupinami a uživateli, protože není dostupná jejich podrobná správa. Toto řešení bylo navrženo s ohledem na použitý způsob řízení přístupu (je popsán v části 6.4.2) a skutečnost, že podrobná správa uživatelů není cílem této práce.

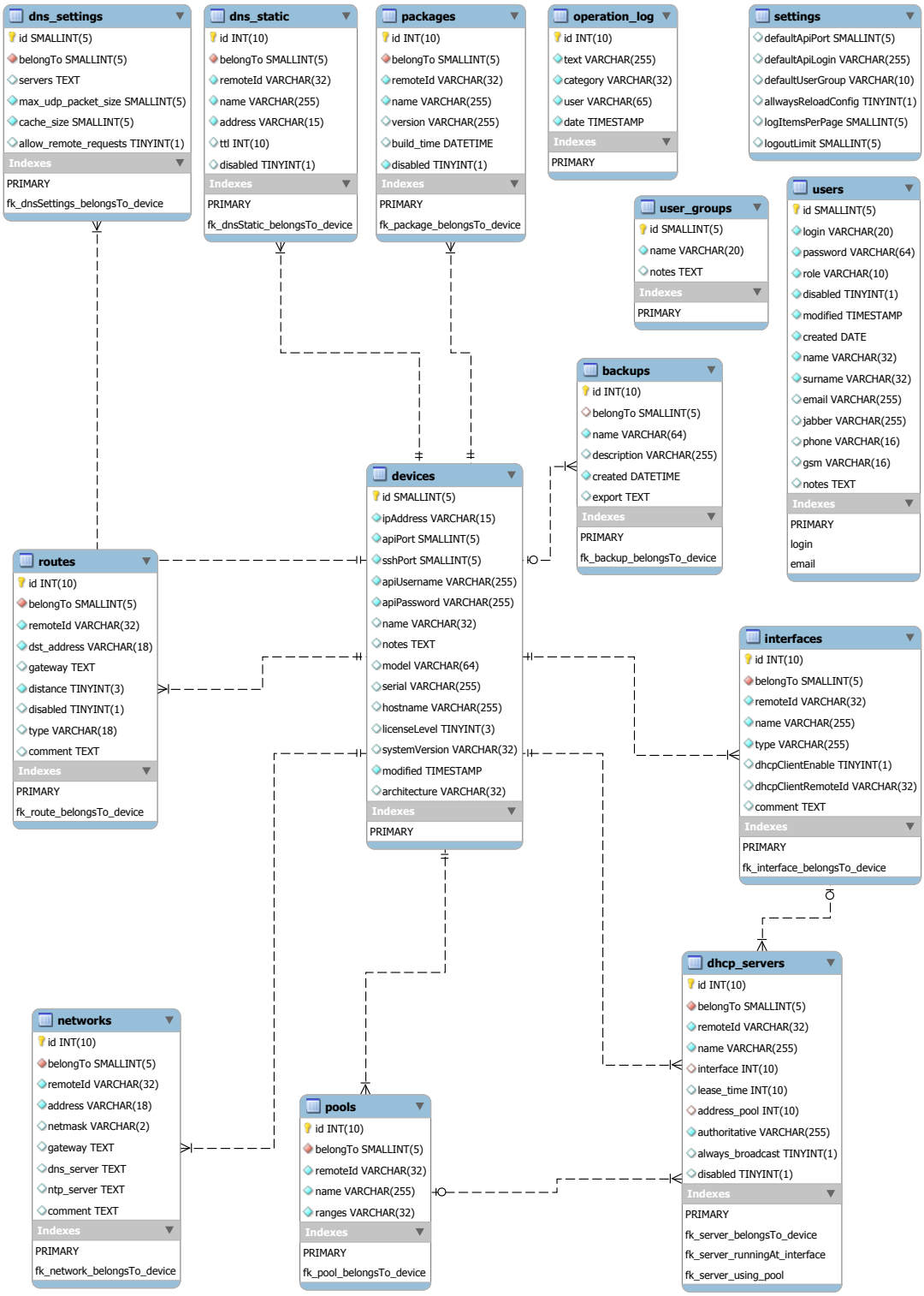
5.4 Zajištění konzistence dat

V souvislosti s prací s daty na straně spravovaných zařízení vznikají dva vzájemně související problémy, které je nutné vyřešit. Patří mezi ně rychlost komunikace, odezva při ní a dále konzistence manipulovaných dat. Je proto nutné najít řešení, které by uspokojivě řešilo oba zmíněné problémy zároveň.

Řešení problému konzistence dat spočívá v použití komunikace v reálném čase. Veškeré čtení dat z koncových zařízení, stejně jako změny v jejich nastavení probíhá hned, jak přichází požadavky na práci s těmito daty. Zásadní omezení tohoto řešení spočívá v rychlosti odezvy při komunikaci se zařízením. Ta může představovat problém zejména na některých pomalejších či dočasně zahlcených linkách. Problém se navíc prohlubuje, uvážíme-li například souběžné operace nad více zařízeními.

Komunikaci se zařízeními lze zrychlit a tím pádem snížit odezvu zavedením tzv. *cache*. Do ní se ukládají data, s nimiž se pracuje a není tak nutné opakovaně stahovat potřebná data ze zařízení. Cenou za snížení odezvy je však nutnost zajistit, aby data v cache byla aktuální a tedy je potřeba provádět jakousi synchronizaci dat mezi systémem a koncovými zařízeními.

V případě mnou navrhovaného systému plní roli cache databáze, do níž jsou potřebná data ukládána. Veškeré operace pak probíhají nad daty v databázi a po ukončení práce jsou změny přeneseny do odpovídajícího koncového zařízení. Synchronizace je jednosměrná, tedy data v databázi mají přednost a přepíší všechny změny v zařízení. Je-li nutné jednorázově aktualizovat data v databázi, lze provést synchronizaci opačným směrem. Dojde tedy k načtení nastavení ze zařízení a data v databázi jsou přepsána.



Obrázek 5.1: Data uchovávaná systémem, reprezentovaná modelem databáze.

Kapitola 6

Implementace systému

Stěžejní kapitola celé práce si klade za cíl zdokumentovat postup implementace jednotlivých částí vyvíjeného systému, jak jsou popsány v části 5.2. Prvně jsou čtenáři přiblíženy technologie a jazyky použité pro implementaci. Následuje stručný popis použité adresářové struktury a architektury *Model-View-Controller* (dále jen MVC), která úzce souvisí se zvoleným frameworkem. Nakonec je popsáno také zabezpečení celé aplikace, zvolený způsob řízení přístupu a detaily implementace jednotlivých funkcí.

6.1 Adresářová struktura

Použitá adresářová struktura respektuje zvyklosti používané v *Nette Framework* (dále jen Nette). Celý systém se skládá z adresáře se soubory aplikace, podpůrných knihoven (například soubory frameworku) a adresáře obsahujícího veřejné soubory, tvořící zejména uživatelské rozhraní. Některé další soubory a adresáře se specifickým významem budou, stejně jako výše uvedené, popsány dále.

Adresář `app` Obsahuje zdrojové kódy aplikace. V podadresáři `presenters` jsou uloženy třídy obstarávající zpracování uživatelských požadavků. Podadresář `models` představuje vrstvu aplikační logiky, resp. třídy zajišťující tuto funkcionalitu. Šablony pro generování výstupu aplikace lze nalézt v adresáři `www`. Speciální význam mají soubory `bootstrap.php` a `config.neon`. První slouží jako zavaděč, obstarávající načtení potřebných knihoven, konfiguraci a spuštění aplikace. Druhý uchovává kompletní nastavení pro různé běhové režimy (vývoj či produkční nasazení).

Adresář `libs` V tomto adresáři a jeho podadresářích jsou uloženy používané knihovny. Jedná se nejen o zdrojové kódy Nette, ale také o třídu pro komunikaci s koncovými zařízeními. Souborem `netterobots.txt` lze vhodně upravit automatické načítání použitých tříd – více informací viz [12].

Adresář `www` Veřejná část aplikace, obsahující především soubory uživatelského rozhraní. Jedná se o kaskádové styly, obrázky a soubory klientského skriptování. Soubor `index.php` spouští aplikaci a následně volá dříve zmíněný `bootstrap.php`. Další obsah tohoto adresáře není nutné popisovat.

Adresář `temp` slouží k odkládání dočasných souborů vzniklých za běhu aplikace. Do `log` jsou poté ukládány záznamy o vzniklých chybách či jiné provozní informace. Pro oba dva

je nutné povolit práva zápisu. Z bezpečnostních důvodů je do některých adresářů zakázán přístup z webového prohlížeče a to pomocí souborů `.htaccess`, resp. `web.config`.

6.2 Použité technologie

S ohledem na koncepci celého systému, je tento tvořen širokou paletou použitých technologií a programovacích jazyků. Několik následujících odstavců se věnuje popisu technologií, souvisejících s uživatelským rozhraním a klientskou částí aplikace. O komponentách serverové části bude řeč později.

Uživatelské rozhraní je tvořeno stránkami v jazycích HTML a JavaScript, vzhledu je dosaženo kaskádovými styly. Hlavním cílem bylo vytvořit ho s důrazem na přehlednost a jeho snadné používání. Jako předloha posloužila šablona *Witti*, která je dostupná pod licencí BSD [1]. Jedná se o jednoduché a přehledné uživatelské rozhraní, určené zejména pro nasazení v administračních systémech. Jejím základním rysem je plně validní XHTML i CSS zdrojový kód. O rozšířené možnosti, jako například automatické odlišený sudých a lichých řádků tabulky, se starají jednoduché skripty v jazyce JavaScript. Samotné Witti obsahuje formátování a styly pro velké množství HTML prvků. Od základních textových elementů až po odstavce, citace, seznamy či obrázky umístěné v textu. Nechybí ani styly pro práci s různými podobami tabulek či možnost vytvářet z obrázků jednoduché galerie. Samozřejmostí jsou také styly pro formátování formulářových prvků a zpráv pro uživatele (chybová hlášení, informace o provedené akci či dotaz na uživatele). Původní podoba šablony byla v průběhu vývoje upravována podle aktuálních potřeb projektu.

Při vývoji se stal serverovým základem software z kategorie *LAMP* [2], konkrétně jeho multiplatformní varianta XAMPP. Hlavní výhodou zvoleného řešení je, že nevyžaduje složitou konfiguraci všech součástí a je možné ho provozovat napříč operačními systémy. V době implementace systému obsahoval balík HTTP server Apache 2.2.21, doplněný o databázi MySQL 5.5.16 a podporu jazyka PHP ve verzi 5.3.8.

Poslední nezbytnou součástí, na které je celá práce postavena je Nette Framework. Jedná se o výkonný a rychle se vyvíjející framework, určený pro rychlý vývoj webových aplikací v jazyce PHP. Nette staví na architektuře MVC (viz. kapitola 6.3), svým přístupem a vlastnostmi aktivně podporuje bezpečnost vytvářené aplikace. Součástí jsou také pokročilé ladící nástroje, umožňující programátorovi pohodlné odhalování chyb, často plynoucích ze značné volnosti jazyka PHP [13]. Výkon aplikace je dále podporován inteligentním načítáním použitých tříd (tzv. *class auto-loading*) a samozřejmě ukládáním používaných součástí do mezipaměti. V produkčním nasazení je možné použít tzv. *minified* verzi, která obsahuje kompletní framework v jednom jediném souboru. Jednoznačnou výhodou je rychlost jeho načtení proti vývojové verzi, skládající se z několika stovek malých souborů. Nevýhodou může být vyšší paměťová náročnost celé aplikace.

Jak již zaznělo v předchozích částech zprávy, komunikace s koncovými zařízeními probíhá pomocí API v systému RouterOS. To bylo podrobně popsáno v kapitole 4 a zde je zmíněno pouze pro úplnost.

[1] Podrobnosti o Witti jsou k nalezení na webu `http://poho.cz/witti`.

[2] LAMP – Linux, Apache, MySQL a PHP (softwarový balík nástrojů pro vývoj webu, vyžadující minimální konfiguraci).

6.3 Architektura MVC

Model-View-Controller (zkráceně MVC) je softwarová architektura, jejíž podstatou je rozdělení struktury aplikace do tří nezávislých komponent. Jednotlivé komponenty spolu komunikují, avšak změny v jedné z nich mají minimální či žádný vliv na zbytek aplikace. Výhodami této architektury je zpřehlednění zdrojového textu aplikace a v budoucnu její snadnější vývoj s možností testovat jednotlivé součásti odděleně. Koncept MVC bývá nejčastěji spojován s webovými aplikacemi a implementuje ho mnoho známých frameworků – např. Nette, Zend a další.

Model Obsahuje datový základ vyvíjené aplikace a veškerou její aplikační logiku. Model sám spravuje svůj vnitřní stav a pomocí známého rozhraní ho nabízí dalším částem aplikace. Jakákoliv akce, kterou může uživatel vyvolat odpovídá akci modelu a jejich voláním dochází ke změnám jeho vnitřního stavu.

View Vrstva starající se o převod dat reprezentovaných modelem do uživateli srozumitelné podoby a o jejich zobrazení. Často tato vrstva používá šablonovací systém a podle šablon sestavuje výsledné webové stránky, resp. transformuje data z modelu.

Controller Představuje řadič, který obsluhuje požadavky pocházející od uživatele. Na základě nich volá související akce modelu a řídí vykreslování získaných dat pomocí pohledu. Výsledkem jeho činnosti může být téměř libovolný soubor, nejen webová stránka.

Z MVC vychází architektura *Model-View-Presenter* (zkráceně MVP), o které se lze dočíst v [14]. V ní *presenter* nahrazuje *controller*, přičemž role obou se mírně liší. Posílena je také role *view* a to o obsluhu událostí, souvisejících s uživatelským rozhraním. Na MVP staví například framework Nette, použitý v této práci.

6.4 Mechanismus přihlašování a řízení přístupu

Vzhledem k tomu, že vzniklý systém bude používat několik skupin uživatelů, bylo nutné vyřešit řízení přístupu jednotlivých skupin ke klíčovým funkcím. Není totiž žádoucí, aby například běžní uživatelé mohli přistupovat k centrálnímu nastavení aplikace či prohlížet systémový protokol.

6.4.1 Autentizace uživatelů

Autentizaci uživatelů při požadavku na přístup do systému obstarává třída `Authenticator` z knihovny *AclLibrary*. Uživatelé jsou autentizováni na základě uživatelského jména, hesla a příznaku, povolujícího jejich účet. Jméno se může skládat pouze z alfanumerických znaků, podtržítka a pomlčky a jeho délka nesmí překročit 20 znaků. Heslo musí mít nejméně 6 znaků, v databázi je poté ukládáno ve formě otisku hashovací funkce. S ohledem na aspekty výkonnostní a bezpečnostní zvolil algoritmus *SHA-256*, doplnění o tzv. *solení hesel*. Jako sůl bylo z praktických důvodů zvoleno uživatelské jméno (viz. [15]).

Přihlášení uživatele je tříkrokové. Uživatel zadá své přihlašovací údaje, které jsou následně předány autentizací službě. Ta z databáze získá informace o tomto uživateli a ověří jeho identitu. Kontroluje se shoda jím zadaného hesla, existence zadaného uživatelského

účtu a také to, jestli není tento účet zakázaný. Nenastal-li v uvedeném procesu problém, je uživatel přesměrován na požadovanou stránku a je uchována jeho identita. V opačném případě je informován o vzniklé chybě a musí pokus o přihlášení opakovat.

6.4.2 Řízení přístupu přes ACL

Access Control List (dále jen ACL) je seznam oprávnění, vztahující se k nějakému objektu, umožňující komplexní řízení přístupu k němu. Základními komponentami ACL jsou *role* (typicky uživatelské skupiny), *zdroje* (chráněný objekt) a *oprávnění* (akce, které mohou být se zdroji prováděny). ACL se dá rozdělit na statické a dynamické. První jmenované uchovává definice základních komponent v souboru a jejich úprava je možná pouze mimo prostředí aplikace. Dynamická varianta pak uchovává svá data typicky v databázi a nastavení ACL lze upravovat přímo za běhu.

Ve své práci jsem využil právě statického ACL. Toto řešení jsem si vybral, protože na výsledný systém nejsou kladeny enormní požadavky, co se řízení přístupu uživatelů týče a také to, že rozsah zdrojů se po dokončení systému prakticky nemění. Dalším z důvodů byla dřívější pozitivní zkušenost se statickou variantou, a z toho plynoucí snadná implementace, oproti řešení dynamickému.

6.4.3 Nastavení oprávnění

V případě této práce jsou za zdroje v ACL považovány jednotlivé chráněné presentery a za oprávnění jejich konkrétní akce. Patřičný *autorizátor* pak nastavuje oprávnění, ke kterým mohou jednotlivé skupiny přistupovat. Všechny skupiny bez rozdílu mají plný přístup k presenterům HomepagePresenter a ErrorPresenter (popsány v částech 6.8 a 6.9). Pro ostatní zdroje platí, že běžní uživatelé (skupina *users*) mohou pouze prohlížet data jednotlivých skupin nastavení a upravovat svůj profil. Nemohou manipulovat s jakýmkoliv nastavením. Členové skupiny *powerusers* mohou pracovat se všemi funkcemi systému, mimo jeho centrální nastavení, prohlížení systémového protokolu a správu uživatelů. Správci (skupina *admins*) nejsou v přístupu jakkoliv limitováni.

6.5 Správa uživatelů (UsersPresenter)

Tento presenter poskytuje funkcionalitu, spojenou s kompletní správou uživatelských účtů. Obsahuje akce pro zobrazení záznamů o všech uživatelích či detailní zobrazení pouze vybraného uživatele. Dále je možné provádět základní manipulaci s daty, tedy přidávat nové účty, upravovat či mazat stávající uživatele. V souvislosti s těmito funkcemi obsahuje *UsersPresenter* také metody, které generují potřebné formuláře a stejně tak metody pro jejich zpracování.

Z pohledu implementace je nejzajímavější částí správy uživatelů univerzální formulář pro jejich vytváření a úpravu. Lze na něm ukázkově demonstrovat velkou sílu použitého frameworku. V závislosti na aktuální akci (vytváření nového či úprava stávajícího uživatele) jsou generovány jednotlivé prvky formuláře a pravidla pro jejich správnou validaci. V případě, že je uživatel vytvářen, systém vyžaduje zadání jeho přihlašovacího jména a nastavení jeho hesla. V případě úpravy uživatele však úprava uživatelského jména není vůbec nabídnuta a vyplnění hesel je povinné pouze, pokud uživatel vyžaduje jejich změnu. Podobným způsobem jsou generovány i další části formuláře.

Popis povinných polí je uživateli odlišen červenou barvou. Veškeré zadávané hodnoty jsou validovány v reálném čase a uživatel tak má okamžitou zpětnou vazbu a může případné chyby ihned opravit. Použitý framework za programátora obstarává také základní validaci formulářů na straně klienta i serveru. Je tak nutné ručně provádět pouze náročnější dodatečnou kontrolu, jak je možné vidět například v případě bran v nastavení směrování v části 6.15. Výsledné rozhraní je k uživateli vstřícné a umožňuje mu upravovat či mazat záznamy nejen z hlavního menu, ale také z kontextových odkazů (například v přehledu uživatelů). To je možné díky metodám, které v případě neznámého identifikátoru záznamu umožní uživateli vybrat záznam, s kterým chce pracovat.

6.5.1 UsersModel

Model pro práci s daty uživatelských účtů zahrnuje všechny potřebné operace a poskytuje transparentní rozhraní pro jejich získávání či manipulaci s nimi. Z databáze tak lze získat informace o všech uživatelích, stejně jako detailní záznam o jednom konkrétním uživateli. Dále je možné přidávat nové uživatele a také editovat, resp. mazat stávající. Nechybí ani podpůrná funkce pro kontrolu existence daného účtu.

6.6 Podpůrné databázové funkce (UtilityModel)

Třída *UtilityModel* obsahuje několik metod pro pohodlnější práci s databází a pro lepší propojení částí *controller* a *model* z architektury MVC. Všechny její metody jsou statické, což umožňuje jejich snadné použití na různých místech aplikace, aniž by bylo nutné vytvářet novou instanci celé třídy.

Metoda `emptyStringsToNull` převádí prázdné řetězce ve vstupním poli na hodnotu `NULL`. Význam má tato metoda zejména před vkládáním dat do databáze, kdy mohou být některé hodnoty, získané z formuláře, prázdné. Použitá databázová knihovna sice umí převádět prázdné řetězce na `NULL`, avšak pouze v případě manipulace se skalární hodnotou. Cílem metody *getSelectboxValues* je snadným způsobem umožnit získávání dat pro naplnění formulářových prvků typu *selectbox*. Na vstupu je očekáván název tabulky, názvy sloupců s identifikátory hodnot a jejich popisem. Volitelně lze také zadat způsob řazení – vzestupně (výchozí) či sestupně. Výstupem je pak pole dvojic hodnot identifikátor záznamu a jeho popis. Poslední je metoda `recordExists`, která slouží k testu existence záznamu v tabulce specifikované jedním ze vstupních parametrů.

6.7 Práce s daty koncových zařízení (MikrotikModel)

Třída *MikrotikModel* je komplexní třída pro práci se všemi tabulkami, které obsahují data, získávaná z koncových zařízení. Při jeho návrhu a implementaci bylo nutné vyřešit problém, spočívající v potřebě přistupovat napříč aplikací k různým částem nastavení koncových zařízení. Proto bylo zvoleno řešení ve formě pouze jednoho modelu pro práci s koncovými zařízeními jako celky. Tento hlavní model není i přes svou rozsáhlost z implementačního pohledu příliš zajímavý a proto bude čtenáři představen pouze stručně.

Hlavním smyslem MikrotikModelu je poskytovat rozhraní pro přístup k nastavení DNS, přes správu nainstalovaných balíčků, až po konfiguraci statického směrování či DHCP. Zároveň slouží k základní evidenci koncových zařízení, s nimiž se v systému pracuje. Metody pracující s daty stejné skupiny jsou tematicky sdružovány a taktéž jsou pro všechny skupiny

dodržovány obdobné jmenné konvence. Nad každou databázovou tabulkou, která reprezentuje některou skupinu funkcí, jsou implementovány základní operace. Je možné přidávat nové, upravovat či mazat stávající záznamy. Samozřejmostí jsou také metody vracející informace o uložených záznamech. Ty podporují jak hromadné získávání informací o více záznamech najednou, tak i poskytování informací o konkrétním záznamu.

V případě práce s nainstalovanými balíčky, správou DNS a statickým směrováním jsou k dispozici i metody pro hromadné vkládání více záznamů najednou, stejně tak i prostředky pro hromadnou aktualizaci již existujících záznamů. Vzhledem k tomu, že úprava více záznamů najednou s sebou nese riziko vzniku nekonzistence databáze (například nových záznamů bude více či méně, než stávajících), je aktualizace takových záznamů řešena odstraněním existujících záznamů a vložením nových, aktuálních, dat. Výhodou je bezproblémové řešení výše popsané situace, nevýhodou pak teoretický problém v plýtvání použitými primárními klíči. Ten by se však projevil pouze při neustálém načítání dat ze zařízení, k čemuž však v běžném provozu není důvod.

6.8 Systémová nástěnka (HomepagePresenter)

Tento presenter plní úlohy vstupní brány do celého systému a je to první stránka, která se zobrazí uživateli po jeho přihlášení. Pokud uživatel má dostatečná práva, je přesměrován rovnou na seznam spravovaných zařízení. V opačném případě je mu zobrazen stručný rozcestník a kontaktní informace na administrátory. Zároveň se jedná o místo, na které je uživatel přesměrován, pokud dojde k chybě, z které se nelze zotavit a není možné mu zobrazit požadovanou stránku. V takovém případě je však zároveň informován o vzniklé situaci zřetelně označenou chybovou zprávou.

HomepagePresenter po svém spuštění zkontroluje, zdali je aktuální uživatel přihlášen a jestli nevypršela platnost jeho přihlášení. Pokud již jeho přihlášení platné není, uloží aplikace jeho požadavek, přesměruje ho na přihlašovací stránku a o vzniklé situaci ho informuje. Mechanismu přihlašování uživatelů je věnována část 6.10. Další úkolem je kontrolovat práva uživatelů. Pokud chce uživatel zobrazit stránku, na níž nemá přístup, dojde k jeho přesměrování na přihlašovací stránku.

Předkem HomepagePresenter je třída *BasePresenter*, jejíž jedinou úlohou je získat informace o uživateli. V případě, že je uživatel přihlášen, je jeho identita (ID, jméno, skupina) předána do šablony pro další použití.

6.9 Obsluha chybových stavů (ErrorPresenter)

Ve vývojovém režimu jsou vzniklé chyby a problémy zobrazovány knihovnou Debugger, která je součástí použitého frameworku. Alternativou pro produkční prostředí je použití speciálně určeného presenteru, který se stará o obsluhu možných chybových stavů. V zaváděcím souboru `bootstrap.php` je nejprve nutné povolit zpracování neodchycených výjimek v produkčním režimu a následně nastavit název presenteru, který se o jejich zpracování bude starat. Příklad je součástí ukázky 6.1.

```
$application->catchExceptions = Environment::isProduction();
$application->errorPresenter  = 'Error';
```

Ukázka 6.1: Nastavení presenteru pro obsluhu chyb v aplikaci.

Související presenter obsahuje pouze jedinou funkci a to `renderDefault`. Ta nejprve zkontroluje, zda vyvolaná výjimka obsahuje některý ze známých stavových kódů protokolu HTTP. Pokud ano, zobrazí se odpovídající chybové hlášení. V opačném případě dojde k zobrazení chyby *500 Internal Server Error* a jejímu zaznamenání pro účely ladění aplikace.

6.10 Přihlašování uživatelů (SignPresenter)

Úlohou *SignPresenter* je správa přihlašování uživatelů do systému a s tím spojené úlohy. Uživatelé se autentizují na základě svého uživatelského jména, voleného při vytvoření účtu a hesla. Uživatelské jméno se může skládat z nejvýše 20 znaků, obsahovat může pouze alfanumerické znaky, podtržítka či pomlčku. Heslo musí mít nejméně 6 znaků. Na znaky, v něm obsažené, omezení kladena nejsou.

Pro potřeby přihlašování vytváří tento presenter formulář, obsahující textová pole pro zadání jména a hesla. Součástí je také zaškrtávací pole, pro zapamatování přihlášení. Formulář je nutné vyplnit a odeslat během 10 minut od načtení stránky, jinak dojde k jeho zneplatnění. Přihlašování je tak chráněno proti zneužití útokem *Cross-site Request Forgery*. Po úspěšném odeslání formuláře je vyvoláno ověření uživatelem poskytnutých autentizačních údajů. Bylo-li přihlášení úspěšné a požadoval-li uživatel některou zabezpečenou stránku, je na ni přesměrován. V opačném případě se mu zobrazí hlavní stránka administrace. Platnost přihlášení se odvíjí od nastavení aplikace. Pokud bylo povoleno zapamatování přihlášení, zůstane uživatel přihlášen 21 dní, v opačném případě je platnost jeho přihlášení nastavena na hodnotu danou nastavením. Výchozím limitem je 60 minut, po nich je uživatel v případě neaktivity odhlášen. Oba časové údaje byly zvoleny na základě experimentálního zjištění.

Skončil-li proces přihlašování neúspěchem, je vzniklá chyba zobrazena uživateli. Ve všech uvedených situacích dojde také k zaznamenání události do systémového protokolu. O mechanismu přihlašování podrobněji pojednává část 6.4.1 na straně 23. Součástí SignPresenter jsou dále už jen akce `login` (obsluha přihlašování) a `logout` (odhlašování uživatele a záznam události do systémového protokolu).

6.11 Autorizace uživatelů (SecuredPresenter)

Představuje základní stavební kámen všech částí aplikace, u nichž je přístup řízen pomocí ACL. Všechny presentery, implementující zabezpečené části aplikace, od něj přímo či nepřímo dědí a tudíž je v nich dostupná kontrola uživatelských oprávnění, která je hlavní náplní *SettingsPresenter*. Ověřování probíhá voláním metody `isAllowed` z třídy `User`. Ta očekává jako své parametry název zdroje (v našem případě název požadovaného presenteru) a oprávnění (zde název akce, která nad zdrojem má být vykonána).

Hlavní roli v tomto presenteru plní funkce `startup`. Ta obsahuje kód, kontrolující přihlášení uživatele a ověřující, zdali jeho požadavek odpovídá přístupovým právům, která mu přísluší. Při každém požadavku na chráněnou stránku je zkontrolováno, zda je uživatel přihlášen a zda nevypršela platnost jeho přihlášení. Pokud by nastala některá z popsaných situací, uživatel je o ní informován, jeho požadavek je uložen a poté je přesměrován na přihlašovací stránku. Zároveň dojde k zapsání této událostí do systémového protokolu. Je-li uživatel korektně přihlášen, je zkontrolováno, jestli je oprávněn přistupovat na požadovanou stránku. Pokud nejsou jeho oprávnění dostatečná, je informován o vzniklé situaci a je mu zobrazena stránka, z které přišel (tzv. *referer*). Není-li jeho referer znám, dojde k přesměrování na hlavní stránku administrace – viz. 6.8.

S ohledem na to, že se jedná o rodičovský presenter většiny částí aplikace, obsahuje některé podpůrné funkce, které by mohlo být nutné používat napříč celým systémem. Jedná se o funkce `rosTimeToSeconds`, která zpracuje časový údaje ve formátu používaném RouterOS a převede ho na odpovídající počet sekund. Zpracování probíhá kontrolou vstupního řetězce a jeho zpracováním odpovídajícím regulárním výrazem, společně s následným převodem na jeho časovou reprezentaci. Dále jsou dostupné funkce `secondsToString` pro převod sekund na lidsky čitelný údaj (formát *0h 0 min 0s*) a také `checkFailResponse`, která kontroluje, zda odpověď získaná z koncového zařízení neobsahuje informace o vzniklých chybách. Tyto tři funkce jsou tak díky dědičnosti dostupné ve všech klíčových částech aplikace.

6.12 Nastavení systému (SettingsPresenter)

Nastavení systému umožňuje správcům uchovat předvolby a výchozí hodnoty některých funkcí. Možnost pracovat s nastavením aplikace skrze webové rozhraní, namísto použití konfiguračních souborů, byla zvolena z důvodu většího uživatelského komfortu a tento presenter vytváří rozhraní pro tuto funkcionalitu. Základem je metoda, vytvářející odpovídající formulář, včetně pravidel pro jeho validaci na straně klienta i serveru. Všechny jeho položky jsou nepovinné. Veškeré číselné položky musí být celá čísla a u jednotlivých polí jsou vhodně omezeny rozsahy hodnoty, které do nich lze zadat. Číslo API portu například respektuje rozsah povolených portů a lze zadat pouze číslo v rozsahu 1024-65535. Tzv. *well-known* porty jsou zakázány. Pole jsou slučována do skupin, na základě toho, o jaká nastavení se jedná.

Po odeslání jsou z korektně odeslaného formuláře získány jednotlivé hodnoty a pomocí metody `saveAllSettings` z příslušného modelu je toto nové nastavení zaneseno do databáze. Po uložení je uživatel informován o výsledku a je přesměrován na hlavní stránku administrace. Dojde-li k jakékoliv chybě, je zobrazena odpovídající informační hláška a uživatel může chyby opravit. Do systémového protokolu je také zaznamenáno, zda uložení nastavení proběhlo úspěšně či nikoliv. Tento presenter obsahuje pouze jedinou akci, která získá z databáze aktuální nastavení a tyto hodnoty vyplní do výše popsaného formuláře. Dojde-li k chybám, je o nich uživatel opět uvědoměn.

V rámci předvoleb lze uložit výchozí uživatelské jméno pro přístup na API a číslo portu, na kterém je dostupná API služba. V souvislosti se správou zařízení lze povolit, aby se při každé úpravě zařízení automaticky načetla i jeho aktuální konfigurace. Dále může administrátor nastavit, do které uživatelské skupiny budou zařazeni nově vytváření uživatelé a po jak dlouhé době nečinnosti budou odhlašováni. V poslední řadě je možné zvolit, kolik záznamů se bude zobrazovat na stránce při prohlížení záznamů systémového protokolu.

6.12.1 SettingsModel

Data nastavení jsou ukládána do samostatné databázové tabulky. Každá konkrétní volba má svůj vlastní sloupec. Kompletní nastavení aplikace je tak možné získat jednoduchým dotazem typu `SELECT`. Tento návrh umožňuje snadné rozšíření, co se množství voleb týče. Dále je také možné snadno povolit každému uživateli jeho individuální nastavení, namísto jednoho centrálního, jak je tomu nyní.

Třída *SettingsModel* poté představuje model, který zpřístupňuje potřebné operace nad databázovou tabulkou, uchovávající nastavení. Základ tvoří metody pro získání kompletního nastavení aplikace a také pro jeho uložení. První jmenovaná načte všechna dostupná nastavení z databáze a vrátí je formou asociativního pole. Druhá poté na svém vstupu

očekává asociativní pole, v němž jsou hodnoty nastavení, které se mají uložit do databáze. Vzhledem k tomu, že žádné nastavení není povinné, jsou před uložením tato vstupní data zkontrolována a prázdné hodnoty jsou nahrazeny speciální hodnotou `NULL`. Dojde-li v některé z uvedených metod k chybě, je vyvolána výjimka a je na volající třídě, resp. metodě, aby tuto výjimku obsloužila.

Poslední metodou, avšak nejčastěji využívanou, je `getSettingByKey`. Ta na svém vstupu očekává název konkrétního nastavení a vrací jeho hodnotu. V případě, že je vyžadována hodnota neznámého nastavení, je vrácena hodnota `NULL`. Použití najde tam, kde je nutné použít uživatelem definované předvolby (například při párování nového zařízení). Tato metoda je statická, což umožňuje její snadné volání bez nutnosti vytvářet instanci celé třídy `SettingsModel`.

6.13 Záznam činnosti uživatelů (LogPresenter)

Tato skupina funkcí poskytuje možnost, jak za běhu aplikace sbírat data o činnosti uživatelů a o akcích, které byly v systému vykonány. Smyslem je zejména umožnit administrátorům kontrolovat, zdali nedošlo k neočekávaným chybám nebo sabotáži ze strany některých uživatelů. Umožňuje také do jisté míry analyzovat bezpečnostní excesy, jako například pokusy o průnik do systému skrze přihlašovací proceduru. Cílem *LogPresenter* je snadná a přehledná manipulace s takto nasbíranými daty.

Pro potřeby zaznamenávání událostí zavádí tento presenter asociativní pole, které slouží pro mapování míst aplikace, kde vznikají události, na lidsky srozumitelné a čitelné názvy kategorií, s nimiž se dále pracuje. Zároveň jsou také definovány konstanty s výchozími hodnotami jednotlivých parametrů událostí. Z důvodu jednotnosti a pohodlné centrální správy jsou popisy událostí uloženy ve formě konstant taktéž v presenteru. Jedná se o jednoduchou alternativu k pokročilým systémům s podporou dynamických událostí a správy nastavení přes grafické rozhraní.

Pro záznam nových událostí do systémového protokolu slouží metoda `logNew`. S ohledem na autonomnost LogPresenter a nutnost zaznamenávat události napříč systémem, byla tato metoda deklarována jako statická. Na počátku jejího volání dojde k validaci vstupních parametrů a v případě nesrovnalostí jsou dosazeny výchozími hodnoty. Popis události je omezen na maximálně 255 znaků, proto jsou všechny znaky nad tento limit před uložením záznamu ořezány. Poté dojde k mapování kódu kategorie na lidsky čitelný název a záznam je uložen do databáze. Všechny případné chyby jsou ignorovány, aby zbytečně nenarušovaly chod aplikace a práci uživatelů.

Zobrazení záznamů systémového protokolu umožňuje uživateli procházet vzniklé události a filtrovat je na základě několika kritérií. Formulářové prvky pro jednotlivé filtry jsou při zobrazení stránky naplněny daty z databáze (viz. následující podkapitola). Při odeslání tohoto formuláře dojde k nastavení hodnot persistentních parametrů [3], podle nichž poté databázový model provádí filtrování záznamů. Akce `show` poté realizuje výše popsané zobrazení záznamů a to tak, že spojuje filtrovací formulář s metodou příslušného modelu a získává potřebná data. Ta jsou poté předána k vykreslení do šablony. Přehled událostí obsahuje také stránkování, tudíž je eliminováno nebezpečí zahlcení klienta či serveru při prohlížení většího množství záznamů. Počet záznamů zobrazených na stránce je možné zvolit v nastavení aplikace.

[3] Persistentní parametr se automaticky přenáší mezi různými akcemi jednoho presenteru.

6.13.1 LogModel

Model obsahuje logiku pro obsluhu databázových operací s tabulkou systémového protokolu. Smyslem systémového protokolu je, aby jeho práce byla transparentní a nenarušovala chod aplikace a nejlépe, aby jeho funkce zůstala uživateli nepovšimnuta. Z tohoto důvodu jsou všechny chyby v modelu tiše zpracovány, ale nejsou již hlášeny dále. Dojde-li například k chybě při vkládání nové události, je tento problém ignorován i s vědomím, že mohlo dojít k nenávratné ztrátě této události.

LogModel umožňuje ukládat do databáze nové události, získávat informace o již existujících záznamech a poskytovat data pro filtrování záznamů při prohlížení systémového protokolu. K tomu slouží metoda `getSelectboxData`, která z databáze získává data ze zadaného sloupce a ta vrací formou asociativního pole. Klíč i hodnota každého prvku pole je totožná, protože se tato hodnota poté používá pro filtrování zobrazených záznamů, jak bude popsáno později. Samozřejmostí je podpora pro práci se sloupci typu `TIMESTAMP` či `DATETIME`, kterou je možné aktivovat druhým parametrem této metody. Výhodou je také skutečnost, že jsou vrácena pouze ta data, která již v databázi existují. Nedochází tak například k zobrazení kategorií, pro něž neexistují v protokolu žádné záznamy.

Stěžejní částí tohoto modelu je metoda `getRecords`, sloužící k získávání informací o zaznamenaných událostech. Podporuje filtrování dat na základě data vytvoření události, její kategorie, či jména uživatele, který ji vytvořil. Jednotlivá kritéria lze mezi sebou libovolně kombinovat, přičemž jsou jednotlivé podmínky mezi sebou spojeny logickým AND. S ohledem na stránkování, popsané v části 6.13, je možné volit také počet zobrazených záznamů (parametr `$limit`) a počet záznamů, které má databáze přeskočit (parametr `$offset`). Aby bylo možné všechny tyto možnosti skloubit, použil jsem techniku skládání SQL dotazu, která je v použité databázové vrstvě podporována. Na začátku metody dojde k uložení základního dotazu do pole a toto pole je následně doplňováno na základě použití jednotlivých filtrů, či pokud je omezen počet zobrazených záznamů. Jakmile je celý dotaz složen, dojde k jeho spuštění a k získání dat, které mu odpovídají. Data jsou vrácena formou asociativního pole. Klíčem je ID konkrétní události, hodnotou potom pole s údaji o ní. V případě, že dojde k chybě, je vráceno prázdné pole a šablona tuto možnost zpracuje při vykreslování výsledné stránky.

6.14 Správa zařízení (DevicesPresenter)

DevicePresenter představuje stěžejní část vyvíjeného systému. Obstarává základní funkcionalitu pro evidenci konfigurovaných zařízení a prostředky pro získávání provozních dat z koncových zařízení. Obsažena je funkcionalita pro základní evidenční operace, jakými jsou párování nových zařízení, úprava a odstraňování stávajících. Spravovaná zařízení lze prohlížet ve stručném přehledu, v němž je také možné vyvolat některé související funkce, či v režimu detailního zobrazení (je popsán dále). Implementována je také základní funkcionalita pro správu a konfiguraci nainstalovaných systémových balíčků. Ty lze zakazovat a povolovat. V souvislosti s těmito akcemi je k dispozici také vzdálený restart spravovaného směrovače. Presenter také obsahuje metody pro přípravu, zpracování a validaci formulářů jednotlivých akcí. Potenciálním problémem je skutečnost, že heslo pro připojení k zařízení přes API musí být uchováváno v čitelné podobě. Z tohoto důvodu není ve formuláři pro správu zařízení heslo zobrazováno skrytě, jak je tomu zvykem. Veškeré pokusy o jeho utajení jsou zbytečné, když bude ve výsledku uloženo čitelně.

Rád bych se krátce zmínil o metodách pro úpravu a odstraňování stávajících záznamů.

Ty svým chováním kladou důraz na uživatelskou přívětivost. Proto umožňují nejen, obvykle dostupné, přímé volání se zadáním identifikátoru zařízení, ale také volání bez tohoto identifikátoru. V takovém případě je uživateli nabídnut výběr dostupných zařízení. Důraz na snadné použití je kladen také v případě zobrazení detailu zařízení. Zobrazeny jsou pak nejen všechny dostupné informace o zařízení, ale také přehled nainstalovaných systémových balíčků s možností jejich správy. Uživateli je samozřejmě dostupné kontextové menu, z kterého má rychlý přístup k jednotlivým částem systému.

Nejrozsáhlejší částí tohoto presenteru je metoda `loadInitData`, sloužící pro načtení aktuálních provozních dat ze spravovaných koncových zařízení. Metoda má definováno pole, obsahující popis dat, která jsou z každého zařízení získána pomocí API v systému RouterOS. Data takto získaná jsou poté upravena v závislosti na tom, o jakou skupinu nastavení se jedná. Dochází například k převodu časových údajů, doplnění hodnot některých příznaků či k úpravě IP adres z CIDR notace [4] do formátu, s jakým systém vnitřně pracuje. Každá skupina nastavení (DNS, směrování apod.) je zpracována samostatně, poté jsou průběžné výsledky spojeny do výsledného vícerozměrného pole a vráceny volající funkci. Výhodou zvoleného řešení je snadné získání kompletního nastavení spravovaných částí koncového zařízení již v průběhu přidávání nového zařízení. Po úspěšném přidání nového zařízení tak máme jistotu, že jsou k dispozici všechna potřebná data. Cenou za toto je nemožnost získávat selektivně pouze některé skupiny nastavení. Při manipulaci s tak velkým množstvím nastavení je nutné důkladně ošetřit možné chybové situace a zajistit konzistenci databáze v případě, že selže ukládání některé skupiny nastavení. V případě, že by tento problém nebyl řešen, zůstávaly by v databázi neplatné záznamy. S ohledem na čistotu řešení jsem zvolil zapouzdření celého vkládání těchto dat do transakce. Ta zaručuje, že v případě chyby bude obnoven předchozí konzistentní stav databáze, aniž by bylo nutné toto provádět ručně.

6.15 Statické směrování (RoutingPresenter)

Úlohou *RoutingPresenter* je podporovat práci se směrováním síťového provozu. S ohledem na požadavky cílového systému a zadání práce bylo vybráno pouze statické směrování. Spravovat nastavení dynamických směrovacích protokolů není v tuto chvíli možné. Uživateli je nabídnuto obdobné rozhraní, jako dříve u dříve popsaného presenteru pro práci s DNS. Opět se v danou chvíli pracuje pouze s jedním zařízením a všechny důležité možnosti jsou uživateli nabízeny kontextově. Pro více detailů lze čtenáři doporučit k přečtení část 6.16.

Zaměření na statické směrování přímo vymezuje funkcionalitu, kterou je potřeba obsáhnout, a také přináší některé problémy, které budou diskutovány později. Základem je možnost procházet existující statické cesty, případně je upravovat či vytvářet nové. Nechybí ani možnost mazání cest či jejich zakazování, respektive opětovné povolení. Doplňkovou funkcí je základní práce se síťovými rozhraními směrovačů. Nezapomnělo se ani na možnost vynuceného načtení konfigurace z koncového zařízení. Lze tak aktualizovat informace o směrování i o síťových rozhraních.

Pro každou cestu jsou uchovávány pouze základní údaje, nutné k jejímu správnému chování. Mezi ně patří cílová IP adresa, metrika dané cesty a typ cesty. V závislosti na typu může či nemusí být vyžadováno zadání výstupní brány. Cesta typu `unicast` vyžaduje zadání alespoň jedné brány, ostatní typy (`blackhole`, `prohibit` a `unreachable`) brány nevyžadují. Tyto tři typy mají speciální význam a ve všech případech je provoz směrovaný

[4] *Classless Inter-Domain Routing* (CIDR) je adresní schéma v IP sítích, podporující beztřídní adresování jednotlivých bodů.

související cesty skrz ně zahazován. Brána může být zadána formou IP adresy či názvem lokálního síťového rozhraní, na které bude odeslán výstupní provoz. Z toho důvodu bylo nutné řešit, jak efektivně umožnit zadávání bran a jak následně tento vstup kontrolovat, vzhledem k možnosti zadávání IP adresy i názvu rozhraní.

Po zralé úvaze bylo zvoleno řešení vycházející ze způsobu zadávání serverů v nastavení DNS. Uživatel tak zadává všechny brány do jednoho pole a jednotlivé záznamy odděluje čárkami. Vstupem tak může být například hodnota `ether1-gateway,192.168.1.1`. Tato je po odeslání zpracována tak, že jsou ponechány pouze platné IP adresy a lokální rozhraní daného koncového zařízení. Ostatní vstupy jsou zahozeny, popř. označeny za chybné a oznámeny uživateli. Blíže je tento mechanismus popsán pseudokódem 6.2. Aby uživatel věděl, která rozhraní může použít, je u formuláře zobrazován jejich seznam.

```
Rozděl_Zadaný_Řetězec_Na_Jednotlivé_Položky;
Inicializuj_Výstupní_Pole;

foreach (Zadané_Položky as Položka) {
  if (Velikost_Výstupu >= 30) {
    break;
  }

  if ((Položka_Je_Prázdná) or (Položka == '(unknown)')) {
    continue;
  }

  if (Položka == '0.0.0.0') {
    Ulož_Chybu;
    continue;
  }

  if ((Je_Položka_IP_Adresa) or (Je_Položka_Síťové_Rozhraní) {
    Přidej_Položku_Do_Výstupního_Pole;
    continue;
  }

  Ulož_Chybu;
}

Nedošlo_K_Chybám ? Vytvoř_Výstupní_Řetězec : Vypiš_Chyby;
```

Ukázka 6.2: Validace zadaných bran u statického směrování.

6.16 Práce s DNS (DnsPresenter)

Třída *DnsPresenter* zpřístupňuje uživatelům systému prostředky pro správu systému DNS. Při svém spuštění dojde automaticky k vytvoření spojení s koncovým zařízením, s nímž se pracuje a také k vytvoření instance modelu, který obstarává přístup k datům v databázi. I zde je použit již zmíněný MikrotikModel, o němž byla řeč v části 6.7.

Nyní bude čtenáři krátce přiblížena filosofie práce s tímto presenterem, která je dále uplatněna také u správy statického směrování a nastavení DHCP. V jednu chvíli se pracuje vždy jen s jedním zařízením. Pokud není při spuštění presenteru zvoleno žádné nebo vybrané zařízení neexistuje, je uživateli zobrazena nabídka, ve které si ho může zvolit. Identifikátor toho, s nímž se pracuje, je uchováván pomocí persistentního parametru. Tímto byl vyřešen problém snadného přenášení identifikátoru mezi voláním jednotlivých akcí presenteru. Výběr pracovního směrovače je možné vždy opětovně vyvolat.

V rámci práce s DNS je implementována široká paleta funkcí, dostupných v systému RouterOS. Je možné upravovat obecné nastavení a to zejména měnit používané DNS servery a spravovat vyrovnávací paměť tohoto protokolu. Tu je možné také jednorázově vyprázdnit, například v případě řešení problémů neplatnými lokálními záznamy. Samozřejmostí je prohlížení statických DNS záznamů, jejich vytváření, úprava či mazání. Všechny události, provedené uživatelem, jsou vždy zaznamenány do systémového protokolu. Funkce pracují s lokálními daty v databázi a vzniklé změny vždy promítají do cílového zařízení. Výjimku tvoří prohlížení vyrovnávací paměti. U té dochází vždy pouze ke stažení aktuálních dat a jejich zobrazení. Tato data se často mění a uchovávat je by bylo nejen plýtvání místem, ale také by mohlo docházet ke nekonzistencím.

Každé zařízení může používat až 30 různých DNS serverů. Z tohoto důvodu bylo nutné vyřešit jejich zadávání. Původně zamýšleným nápadem bylo použít zadávání po jednotlivých serverech s možností dynamického přidávání a odebírání jednotlivých polí. Tato varianta by byla na jednu stranu uživatelsky přívětivá, na druhou stranu by mohla uživatele zdržovat nutností kliknout pro vložení nového pole. Alternativou se stala velmi jednoduchá možnost zadávat jednotlivé servery do jednoho pole a oddělit jejich hodnoty čárkami. Po odeslání formuláře pak dojde k ruční validaci jednotlivých IP adres, odstranění prázdných položek a k dalšímu zpracování.

6.17 Správa DHCP (DhcpPresenter)

Presenter *DhcpPresenter* zajišťuje funkcionalitu, spojenou se správou a nastavením protokolu DHCP a podpůrnými funkcemi s ním spojenými. Stejně jako v případě správy směrování a DNS, je tento presenter koncipován tak, že vždy pracuje pouze s jedním zařízením. Uživateli jsou poté formou kontextových nabídek zobrazovány relevantní funkce, na základě toho, jaké operace zrovna provádí. Cílem tohoto způsobu ovládání je zpřehlednit ovládání rozsáhlé skupiny funkcí.

Implementována je podpora pro práci s rozsahy adres, které přidělují DHCP servery (tzv emphadresní pooly), které lze proházet ve stručném přehledu, vytvářet nové a případně upravovat či mazat stávající. Další důležitou částí práce s protokolem DHCP je možnost definovat nastavení jednotlivých sítí, o nichž servery šíří informace směrem ke klientským stanicím. Lze zadat až dvě výchozí brány, pět DNS serverů a stejný počet serverů služby NTP. Ostatní možnosti nastavení (WINS servery a podobně) byly vynechány s ohledem na to, že jejich použití bývá velmi sporadické. Přestože operační systém RouterOS umožňuje pro každou položku, šířenou skrze DHCP, nastavit až 30 hodnot, ve výsledném systému bylo od tohoto upuštěno. Po konzultaci s vedoucím práce byly zvoleny výše uvedené počty a ty byly shledány jako plně dostačující. Nastavení sítí lze procházet jak v souhrnném přehledu, tak v detailním zobrazení, v němž jsou vidět podrobnosti o nastavení jednotlivých položek. Poslední skupinou funkcí pro základní práci s DHCP je správa serverů. Ty lze vytvářet, měnit jejich nastavení a nepotřebné také mazat. Každý server může být zapnut či vypnut, podle potřeby správců. Opět nechybí ani celkový přehled všech serverů, stejně jako detailní

pohled na konkrétní záznam. V něm jsou uvedeny podobnosti o nastavení použitého rozsahu přidělovaných adres a také informace o síťovém rozhraní, na kterém server běží.

Při prvotním načítání informací o DHCP serverech, které probíhá při přidání nového zařízení do systému, dochází k potenciálnímu problému s konzistencí dat v databázi. To je způsobeno tím, že každý server pro svou činnost vyžaduje nastavení rozsahu přidělovaných adres a případné další nastavení. Tyto hodnoty však v době, kdy jsou informace o serverech načteny, nejsou uloženy v databázi, a tudíž není možné se na ně odkazovat. Z tohoto důvodu je konfigurace DHCP serverů stahována až ve druhé fázi, kdy jsou do databáze již vložena potřebná data. Pro každý server je pak na základě názvu vyhledán identifikátor souvisejícího adresního rozsahu a nastavení sítě. Tyto identifikátory jsou uloženy do databáze, společně s dalšími informace o nastavení serverů. Na konzistenci je nutné pamatovat také v případě opětovného načítání seznamu síťových rozhraní. Zde je uplatněn podobný princip jako výše, je tedy zbytečné ho dále popisovat. Pamatováno je také na situace, kdy dojde o odstranění síťového rozhraní, nad kterým je DHCP server spuštěn, nebo ke smazání rozsahu přidělovaných adres. V takových případech je související identifikátor automaticky nastaven na hodnotu `NULL` a server jako takový samozřejmě zůstává vytvořen.

Základní funkcionalitu v oblasti správy DHCP doplňuje podpora nastavení DHCP klientů. Ty je možné na každém rozhraní podle potřeby zapínat či vypínat. Přehled aktuálního nastavení k dispozici formou stručného přehledu. Poslední užitečnou funkcí je možnost prohlížení aktuálně zapůjčených IP adres. Zdrojová data se vždy načítají přímo z koncového zařízení. Jedná se totiž o informace, které se mohou relativně často měnit, a proto by bylo zbytečné je ukládat do databáze a zvyšovat tak její objem a celkovou zátěž. Jednotlivé zapůjčené adresy je možné rušit a také z nich snadno vytvářet statické záznamy (tzv. *static lease*). Každý takový záznam pak znamená, že MAC adresa konkrétního síťového rozhraní bude vždy dostávat stejnou IP adresu.

6.18 Zálohování zařízení (BackupsPresenter)

Nad rámec povinného zadání práce byl navržen a implementován presenter *BackupsPresenter*. Jeho cílem je nabídnout správcům sítě možnost vytvářet a spravovat zálohy evidovaných zařízení. Uživatelé mohou pro každé zařízení vytvářet libovolný počet záloh. Ty lze poté procházet v souhrnném či detailním přehledu a také již nepotřebné zálohy mazat. V detailním pohledu je možné vidět informace o záloze a zařízení, ke kterému se vztahuje a také její obsah. Správci si také mohou obsah zálohy stáhnout do počítače. V aktuální podobě není možné obnovovat existující zálohy do zařízení, v budoucnu by však tato funkcionalita mohla být přidána.

Z pohledu implementačního je záloha realizována pomocí protokolu SSH. Tento protokol byl zvolen, protože v současné podobě RouterOS API existuje chyba, která při exportu nastavení ignoruje některé jeho části a záloha by pak byla neúplná. Dalším problémem je nemožnost jejího stažení skrze API. Pro získání zálohy by tak bylo nutné používat pravděpodobně protokol FTP. Jeho nevýhodou je nejen přenos hesla v čitelné podobě, ale také fakt, že se jedná o další službu, která na koncovém zařízení musí být spuštěná, a tudíž se jedná o potenciální bezpečnostní hrozbu. Pro práci s SSH byly zvoleny knihovny *phpseclib*, v kterých však musely být provedeny změny v jejich implementaci. Jejich stávající podoba nebyla pro účely použití zcela vyhovující. Upraveno bylo zejména chování v případě chybových stavů.

Samotné zálohování je prováděno neveřejnou metodou `getBackupData`, která očekává na svém vstupu údaje potřebné pro úspěšné navázání spojení – IP adresu, port, přihlašovací

jméno a heslo. Na začátku jsou zadané parametry validovány a poté dochází k připojení ke koncovému zařízení. Bylo-li spojení úspěšné, je v koncovém zařízení proveden příkaz `/export` a jeho výsledek, tedy obsah zálohy, je vrácen volající metodě. Připojení k zařízení a komunikace s ním je již plně v režii výše uvedených knihoven. Dojde-li při zálohování k jakékoliv chybě, je vyhozena odpovídající výjimka a zbytek aplikace se postará o její obsluhu. Všechny významné události, které vzniknou v tomto presenteru jsou, stejně jako události v jiných částech systému, zaznamenány do systémového protokolu.

6.18.1 BackupsModel

Model poskytuje rozhraní, korespondující s funkcionalitou, kterou nabízí BackupsPresenter. Z databáze tak lze získávat souhrnná i detailní data o uchovaných zálohách, ukládat nové či mazat stávající. V souvislosti se zálohami bylo také nutné uspokojivě vyřešit otázku, co se stane se zálohami, pokud dojde ke smazání zálohovaného zařízení. S ohledem na to, že zálohy mohou obsahovat důležitá data, bylo upuštěno od automatického smazání záloh při smazání souvisejícího zařízení. Je-li tedy smazáno zařízení, jeho zálohy jsou uchovány, avšak dojde ke zrušení vazeb mezi příslušným záznamem a zařízením.

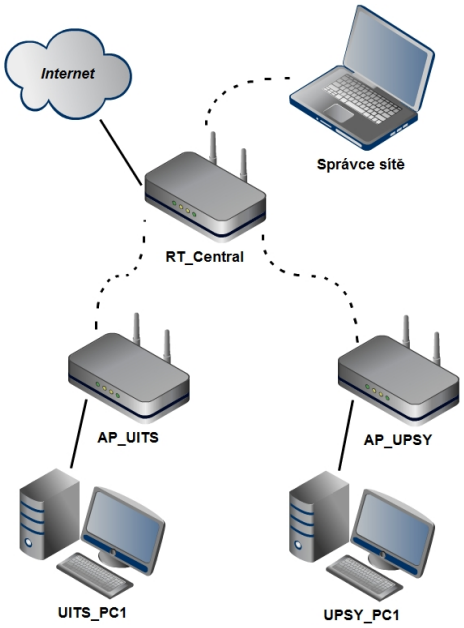
Kapitola 7

Závěr

Cílem této závěrečné kapitoly je demonstrace a shrnutí navrženého systému, jeho výsledné implementace a možností, které poskytuje. Na samém konci je také krátce nastíněno, jakými směry by se mohl v budoucnu ubírat jeho vývoj.

7.1 Demonstrace vytvořeného systému

Tato část se zabývá praktickou ukázkou funkčnosti a možností vytvořeného systému. Celá demonstrace je vedena formou krátkých příkladů, společně s popisem jejich výsledků. Pro otestování byla připravena topologie, skládající se z bezdrátového směrovače, dvou bezdrátových klientů (zařízení *RouterBOARD RB411AH*) a tří počítačů. Bezdrátový směrovač plní roli centrálního bodu, vytváří bezdrátovou síť a je do něj připojen počítač správce sítě. Na tomto počítači také běží předváděný systém. Do vytvořené bezdrátové sítě se také připojují obě zařízení MikroTik a do každého z nich je poté připojen jeden počítač. Na všech zařízeních byly předem povoleny všechny potřebné služby, zejména pak API a SSH. Schéma vytvořené sítě je dostupné na obrázku 7.1.



Obrázek 7.1: Schéma demonstrační topologie.

1. **Přidání všech zařízení MikroTik do evidenčního systému.**
   Nová zařízení budou uloženy do evidence, budou z nich stažena potřebná data a bude možné s nimi dále pracovat.

2. **Zakázání balíčku *ppp* v zařízení AP_UITS a jeho následný restart.**
   Požadovaný balíček bude v daném zařízení zakázán a nebude možné používat jeho funkcionalitu.

3. **Nastavení DHCP (adresní rozsahy, nastavení sítí a servery) v obou zařízeních. Síť UITS bude používat síť 192.168.88.0/24, výchozí brána a DNS server budou mít adresu 192.168.88.1. Síť UPSY bude používat síť 192.168.99.0/24, brána a DNS server bude mít adresu 192.168.99.1.**
   Počítače, připojené k oběma přístupovým bodům získají IP adresu a nastavení sítě. Budou tak schopny komunikovat s jinými prvky v testovací síti. Správné chování je vhodné ověřit nástrojem *ping*.

4. **Změna nastavení DNS v zařízení AP_UITS. K současným DNS serverům bude přidán server 8.8.8.8 a velikost vyrovnávací paměti bude změněna na 4096 KiB.**
   Dané koncové zařízení bude mít nastavený záložní DNS server a bude nastavena nová velikost vyrovnávací paměti pro tento protokol.

5. **Vytvoření statického DNS záznamu v obou přístupových bodech. Doménové jméno bude *central.fit*, cílová adresa bude odpovídat IP adrese směrovače RT_Central.**
   Hlavní směrovač bude pro počítače z obou sítí dostupný pod nastaveným doménovým jménem. Správné chování je vhodné ověřit v internetovém prohlížeči či nástrojem *ping*.

6. **Vytvoření statické cesty v zařízení AP_UPSY. Cesta bude směřovat na IP adresu zařízení RT_Central a bude typu *unreachable*.**
   Hlavní zařízení bude pro počítače ze sítě UPSY nedostupné. Vhodné je ověřit správné chování nástrojem *ping*.

7. **Nastavení statického přidělování IP adresy pro počítač UPSY_PC1.**
   Počítač UPSY_PC1 bude dostávat z DHCP serveru stále stejnou IP adresu.

8. **Vymazání vyrovnávací paměti protokolu DNS v zařízení UPSY.**
   Ve vyrovnávací paměti DNS protokolu zůstanou pouze statické DNS záznamy.

9. **Zakázání dříve vytvořené statické cesty.**
   Směrovač RT_Central je nyní dostupný pro počítače ze sítě UPSY. Vhodné je ověřit správné chování nástrojem *ping*.

10. **Provedení zálohy nastavení zařízení AP_UITS.**
    Aktuální nastavení vybraného přístupového bude uloženo v systému.

7.2 Zhodnocení výsledků

Cíli mé bakalářské práce byly seznámit se s operačním systémem RouterOS, prostudovat možnosti jeho konfigurace, navrhnout systém pro jejich vzdálenou správu a poté ho implementovat. Cílů požadovaných zadáním bylo dosaženo. V prvních třech kapitolách byly

shrnuty poznatky, získané studiem materiálů, určených prvními dvěma body zadání. Na základě takto nabytých znalostí byl v páté kapitole práce popsán návrh informačního systému s webovým rozhraním, jak ho požaduje třetí bod zadání. Tento systém byl následně úspěšně implementován v jazyce PHP s frameworkem Nette a databází MySQL, čímž byl splněn předposlední bod zadání. Na závěr práce byla funkcionalita systému demonstrována na reálné počítačové síti a byla diskutována jeho možná budoucí vylepšení. Tímto byl splněn i poslední bod daný zadáním.

Vytvořený informační systém umožňuje plnohodnotnou konfiguraci statického směrování, protokolu DNS a také DHCP serverů a klientů. Podporuje řízení přístupu uživatelů do systému a protokolování jejich činnosti. Veškeré změny nastavení koncových zařízení jsou do nich zapisovány v reálném čase a pro rychlejší práci jsou ukládány i v lokální databázi. O všech úspěšně či neúspěšně provedených operacích je uživatel vždy informován, aby mohl zjednat nápravu. Nad rámec zadání práce bylo implementováno i zálohování spravovaných zařízení.

Správcům sítě se tak do rukou dostává nástroj, jehož přínos tkví zejména v jednoduchém prostředí a úzkém vymezení dostupné funkcionality. Tento systém, na rozdíl od jiných dostupných nástrojů, nabízí přidanou hodnotu, spočívající v možnosti centrální správy všech zařízení z jednoho místa a v jejich evidenci. Zároveň má perspektivu dalšího vývoje a rozšíření stávající funkcionality. Systém byl také navrhován za účelem praktického využití a v době dokončování této práce je jeho možné budoucí nasazení v jednání.

7.3 Vylepšení do budoucna

S ohledem na cílové nasazení vytvořeného systému se nabízí několik významných rozšíření. Zajisté by se hodilo rozšířit stávající funkcionalitu a nabídnout širší paletu nastavení, která je možné v zařízeních spravovat. Konkrétně by bylo užitečné doplnit podporu práce s nastavením firewallu a správu jeho pravidel. V souvislosti s tím by bylo přínosné integrovat i správu značkování průchozích paketů pomocí funkce *mangle*, která je součástí systému RouterOS. V budoucnu by bylo vhodné doplnit také možnost obnovit lokálně uložené zálohy zpět do koncového zařízení.

Kromě rozšiřování funkcionality, vztahující se k práci s koncovými zařízeními, by mohlo dojít k optimalizaci systému jako celku. Konkrétně by mohla být vylepšena práce s databází a zvýšena efektivita práce se spravovanými zařízeními. V případě potřeby lepšího řízení přístupu by současné ACL mohlo být nahrazeno dynamickou variantou a rozšíření by se mohl dočkat také stávající systémový protokol.

Literatura

[1] Mikrotikls Ltd. *Out customers* [online]. 2007, Modified 2010-11-13 [cit. 27. srpna 2011]. Dostupné na: <http://mikrotik.com/ourcustomers.php>.

[2] Mikrotikls Ltd. *RouterBOARD Price-Performance Comparison* [online]. 2011, aktualizováno 2011-05-19 [cit. 27. srpna 2011]. Dostupné na: <http://routerboard.com/pdf/RouterBOARD_Price_Performance_-Comparison.pdf>.

[3] Mikrotikls Ltd. *RouterBoard.com: RB433* [online]. 2008, aktualizováno 2012-01-30 [cit. 20. března 2012]. Dostupné na: <http://routerboard.com/RB433>.

[4] MikroTik Wiki. *Manual:System/UPS* [online]. 2011, Modified on 12 May 2010 at 13:05 [cit. 1. září 2011]. Dostupné na: <http://wiki.mikrotik.com/wiki/Manual:System/UPS>.

[5] MikroTik Wiki. *Manual:License* [online]. 2011, Modified on 6 July 2011 at 11:59 [cit. 29. srpna 2011]. Dostupné na: <http://wiki.mikrotik.com/wiki/Manual:License_levels>.

[6] MikroTik Wiki. *Manual:API* [online]. 2011, Modified on 2 September 2011 at 08:36 [cit. 13. září 2011]. Dostupné na: <http://wiki.mikrotik.com/wiki/API>.

[7] MikroTik Wiki. *API command notes* [online]. 2011, Modified on 10 August 2011 at 08:35 [cit. 13. září 2011]. Dostupné na: <http://wiki.mikrotik.com/wiki/API_command_notes>.

[8] MikroTik Wiki. *API PHP class* [online]. 2011, Modified on 23 March 2011 at 08:31 [cit. 13. září 2011]. Dostupné na: <http://wiki.mikrotik.com/wiki/API_PHP_class>.

[9] Kabelová, Alena a Dostálek, Libor. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5. vyd. Brno: Computer Press, 2008. 488 s. ISBN 978-80-251-2236-5.

[10] Droms, Ralph a Lemon, Ted. *DHCP příručka administrátora*. 1. vyd. Brno: Computer Press, 2004. 490 s. ISBN 80-251-0130-4.

[11] Gilmore, W. Jason. *Velká kniha PHP 5 a MySQL: kompendium znalostí pro začátečníky i profesionály*. 3. vyd. Brno: Zoner Press, 2011. 736 s. ISBN 978-80-7413-163-9.

[12] Grudl, David. *Auto-loading tříd | Nette Framework* [online]. 2011, Modified: 7.3.2012, 1:03 by Patrik Votoček [cit. 12. března 2012]. Dostupné na: <http://doc.nette.org/cs/auto-loading>.

[13] Vrána, Jakub. *1001 tipů a triků pro PHP*. 1. vyd. Brno: Computer Press, 2010. 456 s. ISBN 978-80-251-2940-1.

[14] Potel, Mike. *MVP: Model-View-Presenter: The Taligent Programming Model for C++ and Java* [online]. 1996 [cit. 12. března 2012]. Dostupné na: <http://wildcrest.com/Potel/Portfolio/mvp.pdf>.

[15] Tichý, Jan. *Solení hesel aneb Sůl nad zlato* [online]. 2007, Upraveno 30.10.2007 [cit. 28. dubna 2012]. Dostupné na: <http://phpguru.cz/clanky/soleni-hesel>.

Příloha A

Obsah přiloženého CD

Přiložené CD obsahuje kompletní zdrojové kódy bakalářské práce a další podpůrné materiály, které s ní souvisejí. Bližší popis struktury CD je uveden níže.

Adresář `dist`	Adresář s produkční verzí systému a textem technické zprávy.
Adresář `etc`	Doplňující soubory, jako například vygenerovaná dokumentace zdrojových textů systému.
Adresář `src`	Zdrojové kódy celé bakalářské práce – výsledného systému i technické zprávy.
Soubor `install.pdf`	Návod na instalaci vytvořeného systému. K dispozici také v příloze B.

Příloha B

Návod na instalaci

Níže uvedené body stručně popisují základní instalaci vytvořeného systému a jeho uvedení do provozu. Instalace je prováděna v prostředí operačního systému Microsoft Windows za použití balíčku XAMPP. Ten je možné stáhnout na webové adrese `http://xampp.org`. Verze nástrojů, které jsou potřeba pro běh aplikace, jsou uvedeny níže. Před spuštěním aplikace je vhodné zkontrolovat konfiguraci webového serveru pomocí nástroje *Requirements Checker*. [1]

Minimální požadavky

- Apache 2.2.21
- MySQL 5.5.16
- PHP 5.3.8
- Knihovny Nette Framework 2.0-beta a Dibi 1.5rc (jsou součástí distribuce aplikace).

Návod na zprovoznění

1. Stáhněte a nainstalujte potřebný software – viz výše.
2. Z přiloženého disku zkopírujte obsah adresáře `dist/application` do připravené složky na vašem webovém serveru (například `bp`).
3. Adresářům `log` a `temp` v adresáři z bodu **2** nastavte přístupová práva tak, aby se do nich dalo zapisovat.
4. Vytvořte databázi a importujte do ní obsah souboru `dist/install_db.sql`. Doporučit lze například nástroj Adminer.
5. V adresáři z bodu **2** si otevřete soubor `app/config.neon`. V sekci `common` upravte část `database` jak je naznačeno níže. Po dokončení úprav soubor uložte.

[1] Bližší informace o nástroji *Requirements Checker* jsou k nalezení na adrese `http://doc.nette.org/cs/requirements`.

```
driver: mysql
host: ADRESA_SERVERU
database: JMENO_DATABAZE
username: UZIVATELSKE_JMENO
password: HESLO
profiles: false
charset: utf8
```

6. V internetovém prohlížeči otevřete adresu, na které je aplikace nainstalována (například `http://localhost/bp`).

7. Přihlaste se do systému uživatelským jménem **admin** a heslem **admin1234**. Po úspěšném přihlášení toto výchozí heslo změňte.

Příloha C

Metriky kódu

S výjimkou celkové velikosti zdrojových kódů jsou do metrik zdrojových souborů zahrnuty pouze vlastní části programu, nikoliv podpůrné knihovny třetích stran. Jedná se tak o statistiky vytvořené z presenterů, modelů a šablon, doplněné o několik dalších doplňujících souborů.

Počet zdrojových souborů	89 souborů
Počet řádků zdrojového textu na soubor	≈ 123 řádků
Počet řádků zdrojového textu celkem	10 875 řádků
Velikost vlastních zdrojových souborů	426,5 kB
Celková velikost zdrojových souborů	3061 kB
Počet presenterů	13 presenterů
Počet modelů	7 modelů
Počet ostatních tříd	6 tříd